高等计算机体系结构
第五讲: 多周期和流水线

栾钟治
北京航空航天大学 计算机学院 中德联合软件研究所
2021-03-26

1

提醒：作业

- 作业 1
  - 已截止
  - MIPS 、ISA 基本概念，基本的性能分析评价
- 作业 2
  - 今天发布，4月9日上课前截止
  - 单周期与多周期微体系结构
- 作业 3
  - 4月9日发布
  - 流水线

2

2

提醒：实验 1

- 已发布，4月16日提交
  - 用Logisim设计1个7指令单周期MIPS CPU
- 学习MIPS ISA

3

3

阅读材料

- Patterson & Hennessy's *Computer Organization and Design: The Hardware/Software Interface* （计算机组成与设计：软硬件接口）
  - 附录 D
  - 第四章 (4.5-4.8, , 4.9-4.11)
- 选读
  - Maurice Wilkes, "The Best Way to Design an Automatic Calculating Machine," Manchester Univ. Computer Inaugural Conf., 1951.
  - Smith and Sohi, "The Microarchitecture of Superscalar Processors," Proceedings of the IEEE, 1995
  - Patt & Patel's *Introduction to Computing Systems: From Bits and Gates to C and Beyond* （计算机系统概论）
    - 附录C ：LC-3b ISA及微体系结构

4

4

回顾： “处理指令”的步骤

- ISA 抽象地说明给定一条指令和A，A’应该是什么
  - 定义一个抽象的有限态机
    - 状态 = 程序员可见的状态
    - 次态逻辑 = 指令执行的规范
  - 从 ISA 的视角，指令执行的过程中A和A’之间没有“中间状态”
    - 每条指令对应一个状态转换
- 微体系结构实现 A 向 A’的转换
  - 有很多种实现方式的选择
  - 我们可以加入程序员不可见的状态来优化指令执行的速度：每条指令有多个状态转换
    - 选择 1：A → A’（在一个时钟周期内完成 A 到 A’ 的转换）
    - 选择 2：A → A+MS1 → A+MS2 → A+MS3 → A’(使用多个时钟周期完成 A 到 A’的转换)

5

5

回顾：单周期 vs. 多周期

- 单周期的机器
  - 每条指令执行需要一个时钟周期
  - 所有状态的更新在指令执行结束的时刻完成
  - 劣势：最慢的指令决定时钟周期的长度 → 时钟周期时间长
- 多周期的机器
  - 指令处理分到多个周期/阶段中完成
  - 指令执行过程中可以更新状态
  - 但是体系结构状态的更新只能在指令执行结束的时刻完成
  - 与单周期相比的“优势”：最慢的“阶段”决定时钟周期长度
    - 单周期和多周期在微体系结构层面都遵从冯诺依曼结构

6

6

回顾：观察指令处理的另一个视角

- 指令将数据(AS)转换成数据 (AS’)
- 由功能单元完成转换
  - “操作”数据的单元
- 需要有人告诉这些单元对数据做什么操作
- 一个指令处理的引擎由两部分组件构成
  - 数据通路：由处理和转换数据信号的硬件部件组成
    - 操作数据的功能单元
    - 存储数据的存储单元（比如寄存器）
    - 使数据流能够流入功能单元和寄存器的硬件结构（比如连线和多路选择器）
  - 控制逻辑：由决定控制信号的硬件部件组成，这些控制信号决定了数据通路上的部件会如何操作数据

7

7

回顾：单周期vs. 多周期:控制&数据

- 单周期的机器：
  - 数据信号操作的同时产生控制信号（在同一个时钟周期内起作用）
  - 与一条指令相关的所有事情都发生在一个时钟周期内
- 多周期的机器：
  - 下一个周期需要的控制信号可以在前一个周期就产生
  - 数据通路上的延迟可以和控制处理的延迟重叠

8

8

回顾：初步的性能分析

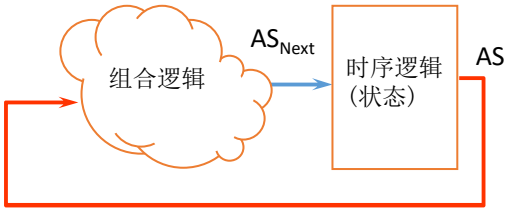
- 指令执行时间
  - {CPI} x {clock cycle time}
- 程序执行时间
  - 所有指令的[{CPI} x {clock cycle time}]之和
  - {指令数} x {平均 CPI} x {clock cycle time}
- 单周期微体系结构的性能
  - CPI = 1
  - Clock cycle time 长
- 多周期微体系结构的性能
  - CPI = 每条指令不同
    - 平均 CPI → 希望能很小
  - Clock cycle time 短

现在，我们有两个独立的自由度可以优化

9

回顾：单周期微体系结构的实现

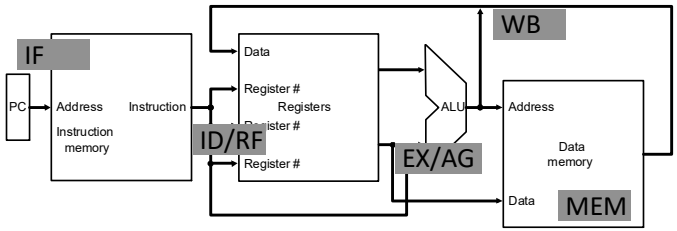
- 单周期的机器



10

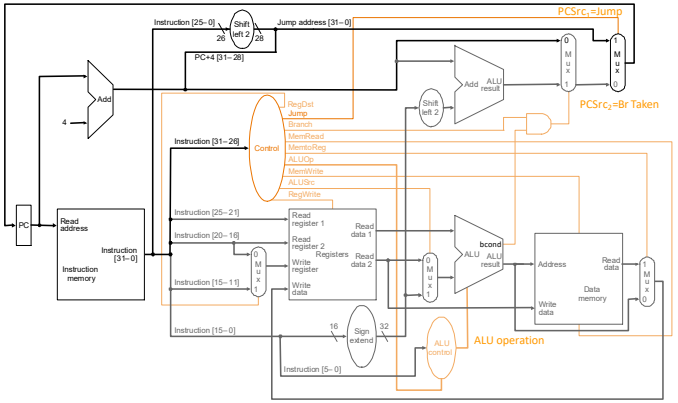
回顾：指令处理

- 5个一般步骤 (Patterson & Hennessy's Book)
  - 取指令 (IF)
  - 指令译码和取寄存器操作数 (ID/RF)
  - 执行/计算内存地址 (EX/AG)
  - 取内存操作数 (MEM)
  - 存储/写回结果 (WB)



11

回顾：完整的数据通路



忽略JAL, JR, JALR

12

回顾：椭圆形的“Control”圈圈里是什么?

- 组合逻辑→ 硬连线控制
  - 思路：基于指令用组合逻辑生成控制信号
- 时序逻辑→ 时序/微程序控制
  - 控制存储
  - 思路：用一个存储结构保存指令的控制信号

13

13

回顾：单周期微体系结构：分析

- 每条指令执行占用1个时钟周期
  - CPI (Cycles per instruction) = 1
- 每条指令执行的时间受限于执行最慢的那条指令
  - 即使很多指令不需要执行那么长时间
- 微体系结构中的时钟周期长度由完成最慢的指令所需时间决定
  - 处理最慢指令的时间决定了关键路径的设计

14

14

回顾：最慢的指令流程是什么?

- 存储器不是理想的
- 如果有时候访存要花费100ms怎么办?
- 让简单的寄存器加或者无条件转跳花上和访存操作一样的100ms+的时间有意义吗?
- 另外，如果处理一条指令需要不止一次访存该怎么办?
  - 什么指令需要?
  - 是否提供了多个内存端口?

15

15

单周期微架构:复杂性

- 人为因素
  - 所有指令都和最慢的指令一样慢
- 低效
  - 所有指令都和最慢的指令一样慢
  - 必须为所有指令提供最坏情况下的资源
  - 对于一条指令执行周期中在不同阶段会访问同一个资源的情况，必须为该资源提供“副本”
- 不一定是实现ISA的最简单方法
  - REP MOVS, INDEX, POLY等指令的单周期实现?
- 不容易优化/提升性能
  - 对通常情况(普通指令)做优化不起作用
  - 任何时候都要优化最坏的情况

16

16

微体系结构设计原则

- 关键路径设计
  - 找到时延最大的组合逻辑，尽可能的减小它的时延
- 基本（典型）设计
  - 在重要的地方花时间和资源
    - 提升机器设计目标要求的应有能力
  - 通常情况 vs. 特殊情况
- 平衡设计
  - 平衡流过硬件部件的指令/数据流
  - 平衡完成工作所需要的硬件
- *单周期微体系结构是如何遵循这些原则的？*

17

17

多周期微体系结构

18

18

多周期微体系结构

- 目标：使每一条指令的执行只（大致）花费它该花费的时间
- 思路
  - 时钟周期的决定独立于指令处理时间
  - 每条指令需要花费多少时钟周期
    - 一条指令执行过程中会有多次状态转换
    - 每条指令的状态变换是不同的

19

19

回顾：“处理指令”的步骤

- ISA 抽象地说明给定一条指令和A，A’应该是什么
  - 定义一个抽象的有限态机
    - 状态 = 程序员可见的状态
    - 次态逻辑 = 指令执行的规范
  - 从 ISA 的视角，指令执行的过程中A和A’之间没有“中间状态”
    - 每条指令对应一个状态转换
- 微体系结构实现 A 向 A’的转换
  - 有很多种实现方式的选择
  - 我们可以加入程序员不可见的状态来优化指令执行的速度：每条指令有多个状态转换
    - 选择 1：A → A’（在一个时钟周期内完成 A 到 A’ 的转换）
    - 选择 2：A → A+MS1 → A+MS2 → A+MS3 → A’（使用多个时钟周期完成 A 到 A’的转换）

20

20

多周期微体系结构

AS = 指令执行之前程序员可见的体系结构状态

第1步：在一个时钟周期内处理一部分指令

第2步：在下一个时钟周期内处理一部分指令

…

AS' =指令执行之后程序员可见的体系结构状态

21

21

多周期设计的好处

- 关键路径设计
  - 可以独立地针对每条指令的最糟糕情况来优化关键路径
- 基本(典型)设计
  - 可以通过优化执行“重要”指令（占用大量执行时间）所需的状态数来达到需要的效果
- 平衡设计
  - 不需要提供比实际需求更多的资源或能力
    - 一条指令需要多次使用资源“X”并不意味着需要多个“X”
    - 使硬件更高效：一条指令可以多次***重用***硬件部件

22

22

性能分析

- 指令执行时间
  - {CPI} x {clock cycle time}
- 程序执行时间
  - 所有指令的[{CPI} x {clock cycle time}]之和
  - {指令数} x {平均 CPI} x {clock cycle time}
- 单周期微体系结构的性能
  - CPI = 1
  - Clock cycle time 长
- 多周期微体系结构的性能
  - CPI = 每条指令不同
    - 平均 CPI → 希望能很小
  - Clock cycle time 短

有两个独立的自由度可以优化

23

23

CPI vs. 主频

- CPI vs. 时钟周期长度
- 互相矛盾
  - 对一条指令来说，减少一个就会增加另一个
  - 为什么?
- 多条指令并发处理可以使平均CPI被平摊/减小
  - 同一个时钟周期被用来处理多条指令
  - 例如：流水线，超标量等

24

24

多周期微体系结构
近距离观察

25

25

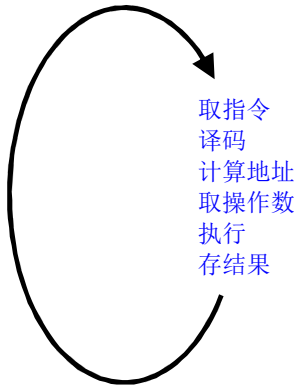
如何实现多周期?

- Maurice Wilkes, "The Best Way to Design an Automatic Calculating Machine," Manchester Univ. Computer Inaugural Conf., 1951.
- 微码/微程序控制的概念
- 实现
  - 可以按照描述状态之间序列的有限状态机来实现"指令处理"的步骤，最终状态机回到"取指令"状态
  - 状态由控制信号推定
  - 下一个状态的控制信号由当前状态决定

26

26

指令执行周期



27

27

基本的多周期微体系结构

- 指令执行周期被划分为多个"状态"
  - 指令执行周期的每个阶段可以拥有多个状态
- 多周期微体系结构通过状态到状态的序列处理指令
  - 某个状态下机器的行为由该状态下的控制信号决定
- 整个处理器的行为可以被定义成一个有限状态机
- 在某个状态(时钟周期)中，控制信号控制
  - 数据通路如何处理数据
  - 如何为下一个时钟周期生成控制信号

28

28

微程序控制相关术语

- 与当前状态相关的控制信号
  - 微指令
- 从一个状态过渡到另一个状态的动作
  - 决定下一个状态以及下一个状态的微指令
  - 微序列（生成）
- 控制存储（器）为每一个可能的状态存储控制信号
  - 为整个有限状态机存储微指令
- 微序列（控制）器决定下一个时钟周期（下一个状态）将会用到的控制信号集合

29

29

在一个时钟周期里发生了什么?

- 对当前状态控制的控制信号(微指令)
  - 在数据通路中推进
  - 为下一个周期生成控制信号（微指令）
- 数据通路和微序列器并发操作
- 问题：为什么不在当前周期生成当前周期需要的控制信号?
  - 会使时钟周期延长
  - 为什么?

30

30

简单的LC-3b控制和数据通路

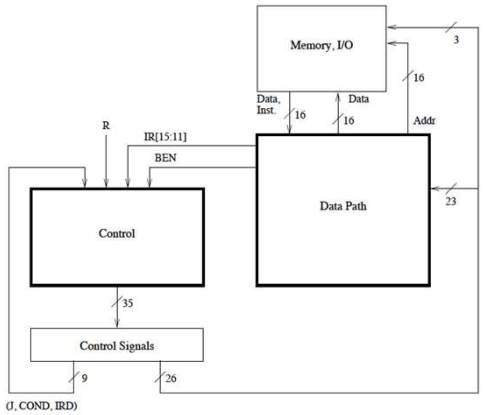


Figure C.1: Microarchitecture of the LC-3b, major components

31

31

什么决定了下一个周期的控制信号?

- 当前时钟周期发生了什么
  - 流入“Control”框的9根线
- 被执行的指令
  - 来自数据通路的IR[15:11]
- 不管分支条件是否满足，如果执行的指令是分支
  - 来自数据通路的BEN（1 bit）
- 不管访存操作是否在本周期完成，只要有访存操作在处理
  - 来自存储器的R（1 bit）

32

32

LC-3b多周期处理的状态机

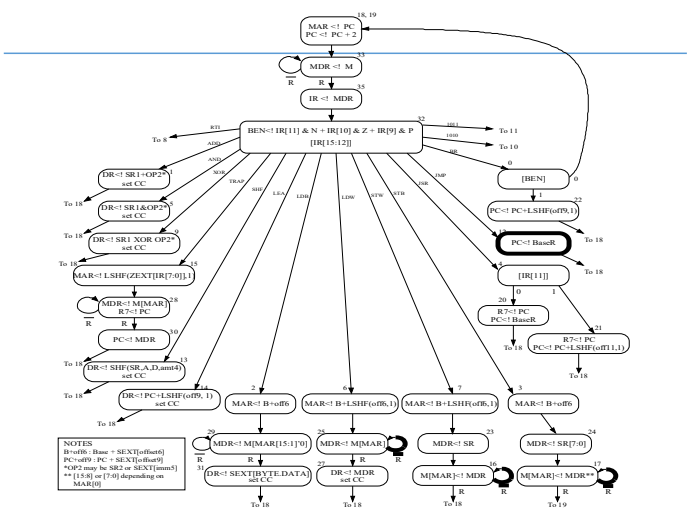
- LC-3b微架构的行为由以下因素决定
  - 35个控制信号和7位由数据通路连入控制逻辑的信号
- 35个控制信号完整描述了控制结构的状态
- LC-3b的所有行为都可以描述为状态机——一个有向图
  - 结点(关联到每个状态)
  - 弧(代表一个状态到另一个状态的流)

33

LC-3b状态机

- Patt & Patel, 附录 C, 图 C.2
- 每个状态描述必须是唯一的
  - 通过*状态变量*
- LC-3b状态机有31个不同的状态
  - 由6个状态变量编码
- 例如
  - 状态18,19对应指令处理周期的开始
  - 取指阶段：状态18, 19 → 状态33 → 状态35
  - 译码阶段：状态32

34



35

关于LC-3b状态机的几个问题

- 最快的指令执行需要多少个时钟周期?
- 最慢的指令执行需要多少个时钟周期?
- 什么决定了时钟周期?
- 这是个摩尔型状态机还是米里型状态机?

36

简单的LC-3b控制和数据通路

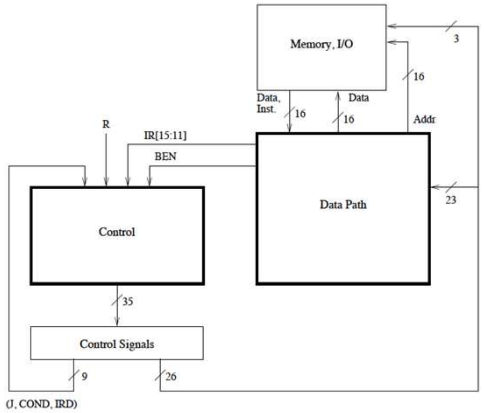


Figure C.1: Microarchitecture of the LC-3b, major components

37

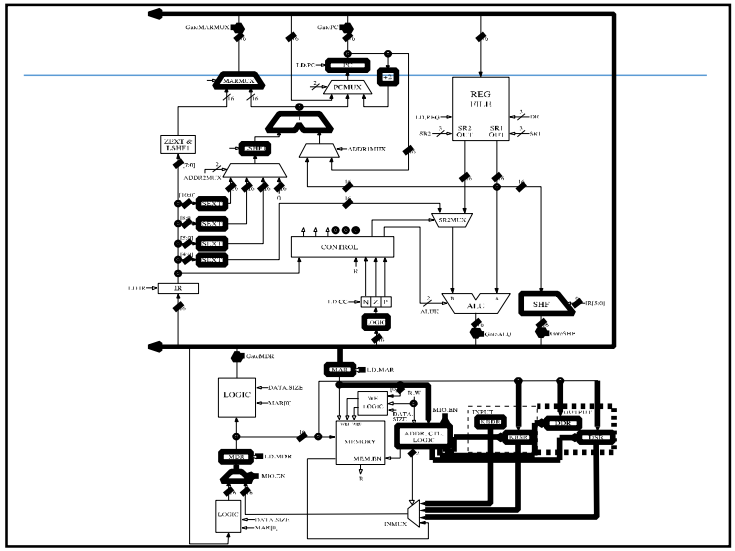
37

LC-3b数据通路

- Patt & Patel, 附录 C, 图 C.3
- 单总线数据通路设计
  - 任何时候只能有一个数值被放上总线(选通并使用总线)
  - 优点：硬件成本低，只有一条总线
  - 缺点：降低了并发性 - 如果指令需要因为两件不同的事使用总线两次，需要在不同的状态中完成
- 26个控制信号决定了一个时钟周期内数据通路上发生什么
  - Patt & Patel, 附录 C, 表 C.1

38

38



39

39

Signal Name	Signal Values	
LD.MAR/1:	NO, LOAD	
LD.MDR/1:	NO, LOAD	
LD.IR/1:	NO, LOAD	
LD.BEN/1:	NO, LOAD	
LD.REG/1:	NO, LOAD	
LD.CC/1:	NO, LOAD	
LD.PC/1:	NO, LOAD	
GatePC/1:	NO, YES	
GateMDR/1:	NO, YES	
GateALU/1:	NO, YES	
GateMARMUX/1:	NO, YES	
GateSHF/1:	NO, YES	
PCMUX/2:	PC+2	;select pc+2
	BUS	;select value from bus
	ADDER	;select output of address adder
DRMUX/1:	11.9	;destination IR[11:9]
	R7	;destination R7
SR1MUX/1:	11.9	;source IR[11:9]
	8.6	;source IR[8:6]
ADDR1MUX/1:	PC, BaseR	
ADDR2MUX/2:	ZERO	;select the value zero
	offset6	;select SEXT[IR[5:0]]
	PCoffset9	;select SEXT[IR[8:0]]
	PCoffset11	;select SEXT[IR[10:0]]
MARMUX/1:	7.0	;select LSHF(ZEXT[IR[7:0]],1)
	ADDER	;select output of address adder
ALUK/2:	ADD, AND, XOR, PASSA	
MIO.EN/1:	NO, YES	
R.W/1:	RD, WR	
DATA.SIZE/1:	BYTE, WORD	
LSHF1/1:	NO, YES	

Table C.1: Data path control signals

40

40

关于LC-3b 数据通路的几个问题

- 在数据通路中是如何做到根据状态机实现取指令的?
- 选通和载入有什么不同?
- 这个设计是最节省硬件的吗?

41

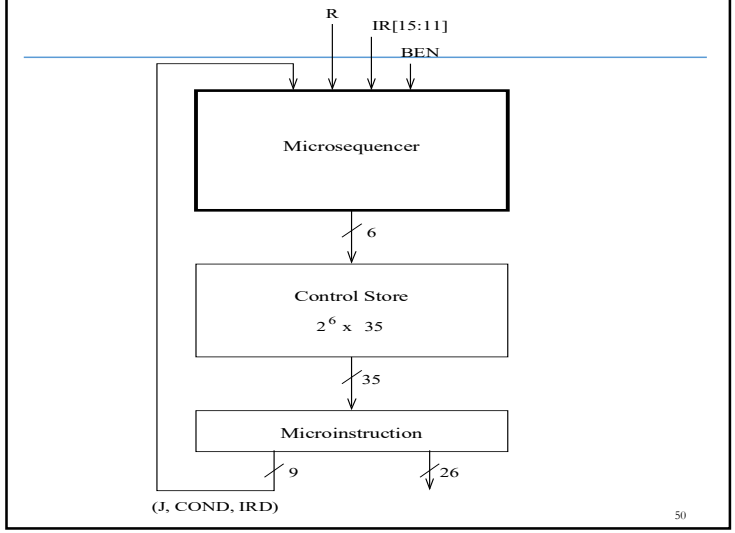
41

LC-3b 微程序设计控制结构

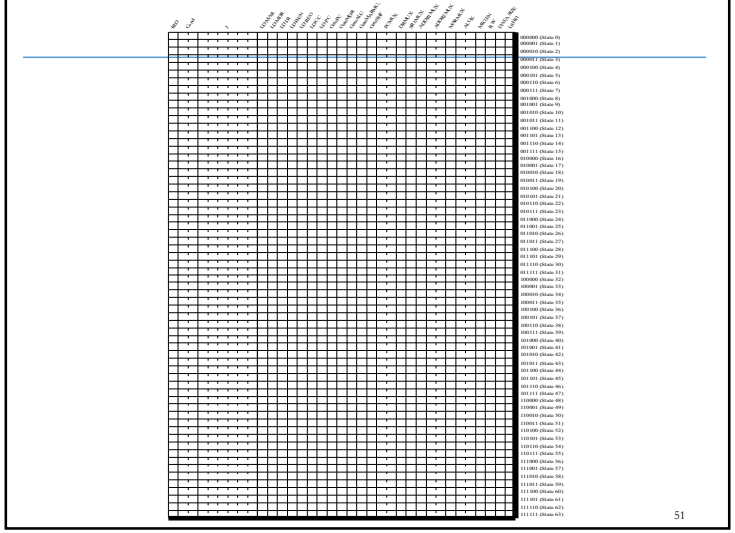
- Patt & Patel, 附录 C, 图 C.4
- 三个组件:
  - 微指令, 控制存储, 微序列（控制）器
- 微指令: 26个控制数据通路，9个决定下一个状态
- 每个微指令存储在控制存储（特殊的存储结构）的特定位置
- 特定位置: 对应微指令的状态地址
  - 每个状态对应一条微指令
- 微序列（控制）器决定下一条微指令的地址（下一个状态）

42

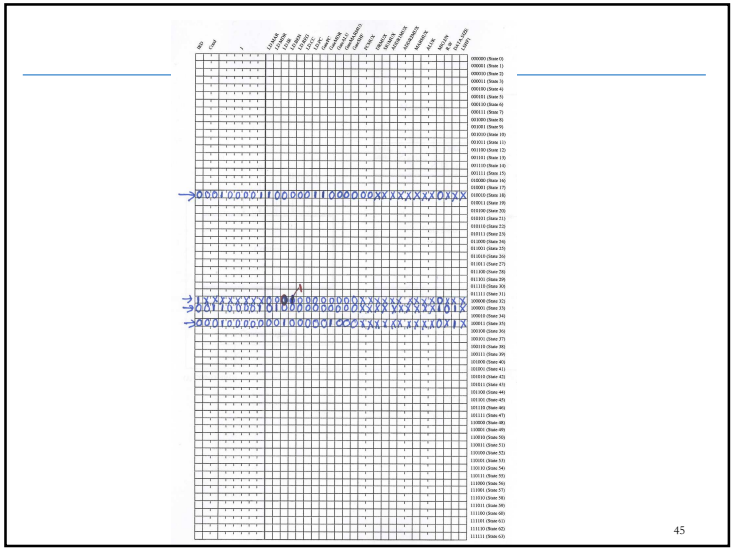
42



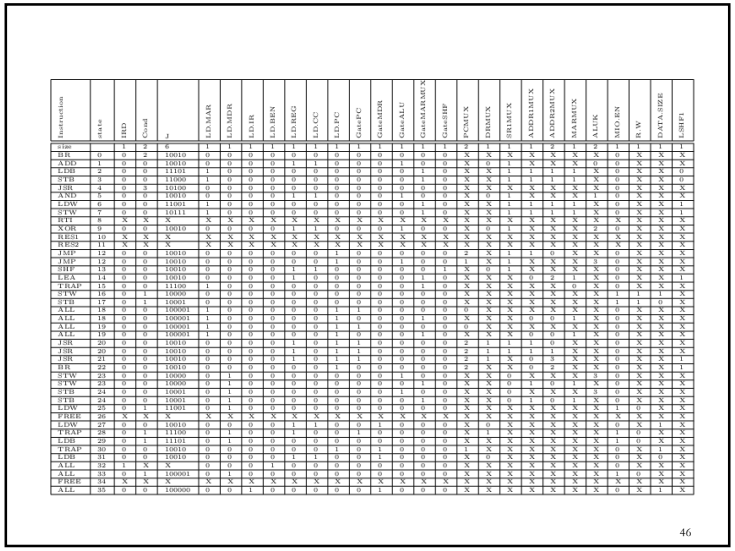
43



44



45



46

LC-3b 微序列（控制）器

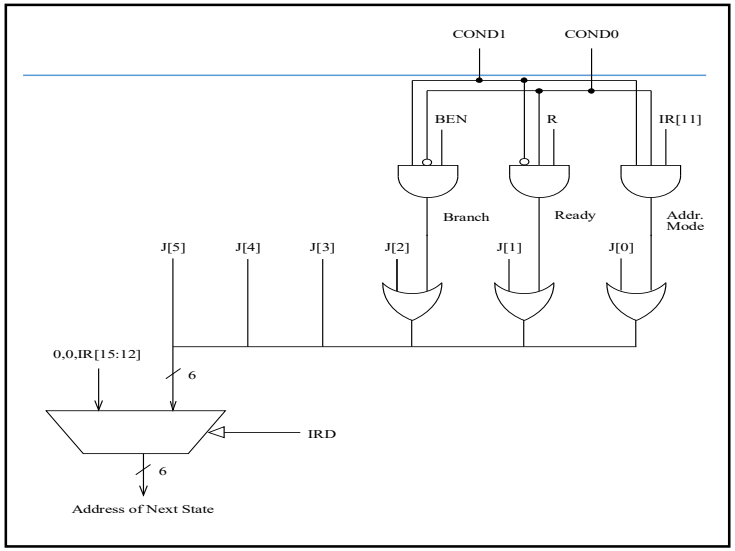
- Patt & Patel, 附录 C, 图 C.5
- 微序列器的目的是决定下一条微指令的地址(下一个状态)
- 下一个地址取决于 9 个控制信号

Signal Name	Signal Values	
J/6:		
COND/2:	$COND_0$	;Unconditional
	$COND_1$	;Memory Ready
	$COND_2$	;Branch
	$COND_3$	;Addressing Mode
IRD/1:	NO, YES	

Table C.2: Microsequencer control signals

47

47



48

关于微序列（控制）器的几个问题

- IRD信号什么时候生效?
- 如果一条非法指令被译码会发生什么?
- 条件(COND)位是用来做什么的?
- 延迟可变存储如何处理?
- 如何对状态编码?
  - 使状态变量数最少
  - 从16路分支开始
  - 然后根据COND位确定约束表和状态

49

49

关于控制存储的几个问题

■什么控制信号能够被存入控制存储?

■什么控制信号只能由硬连线逻辑生成?

❑ 什么信号必须在数据通路中处理才能得到?

50

50

延迟可变存储

- Ready信号(R)使得存储器读写能够正确的执行
  - 例如: 状态33向状态35转变由存储器准备好后生成的R信号控制
- 在单周期微体系结构中是如何做的?

51

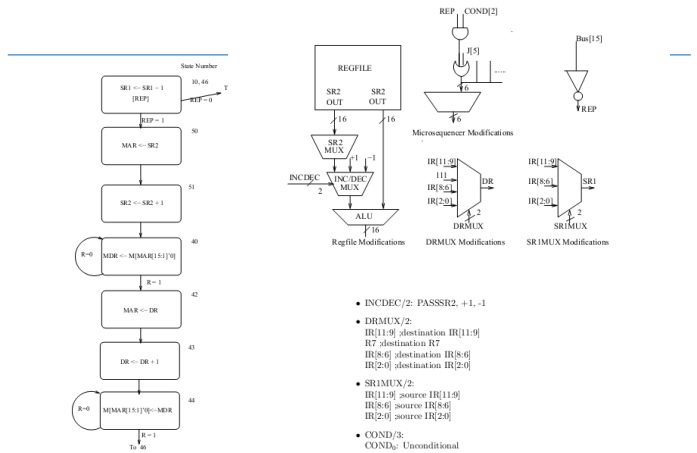
51

关于微序列器的高级问题

- 如果机器出现中断会发生什么?
- 如果指令产生异常会怎么样?
- 如何使用这种控制结构实现一条复杂指令?
  - 考虑 REP MOVS

52

52



53

抽象的力量

- 控制存储的微指令概念使得硬件设计者具有一种新的抽象：微程序设计
- 设计者可以将任何希望的操作翻译成微指令序列
- 设计者只需要提供
  - 实现目标操作所需的微指令序列
  - 具有正确驱动微指令序列能力的控制逻辑
  - 其它必须附加的数据通路控制信号(如果操作不能翻译成已有的控制信号)

54

54

其它：内存中的对齐矫正

- 访存对齐
- LC-3b 有字节 load 和 store 指令，可以不按照字边界移动数据
  - 对程序员/编译器很方便
- 硬件如何保证读写的正确性?
  - 状态 29 - LDB
  - 状态 24 和 17 - STB
  - 额外的逻辑处理未对齐的访问

55

55

其它：内存映射I/O

- 地址控制逻辑决定访存指令的地址是内存还是I/O设备
- 相应地驱动内存或I/O设备并且设置多路选择器
- 有些控制信号不能保存在控制存储中
  - 依靠地址

56

56

微程序控制的优点

- 通过控制数据通路（用序列器），可以用非常简单的数据通路实现强有力的计算
  - 高级ISA翻译成微码（微指令序列）
  - 微码使得用最简单的数据通路***仿真***ISA成为可能
  - 微指令可以被看作是用户不可见的ISA
- 使ISA很容易扩展
  - 可以通过改变微码支持新的指令
  - 可以通过简单微指令的序列来支持复杂的指令
- 如果可以把任意指令序列化，那么也能够把任意“程序”序列化成微程序序列
  - 在微码中需要一些新的状态（如：循环计数器）来序列化更复杂的程序

57

硬件升级

- 对微码升级/打补丁的能力（处理器发货之后）
  - 不用更换处理器就可以增加新的指令!
  - “修复”硬件实现的缺陷
- 例如
  - IBM 370 Model 145：微码存储在主存中，可以在重启之后升级
  - IBM System z：与 370/145类似
    - Heller and Farrell, “Millicode in an IBM zSeries processor,” IBM JR&D, May/Jul 2004.
  - B1700 微码可以在处理器运行时更新
    - 用户可微编程的机器!

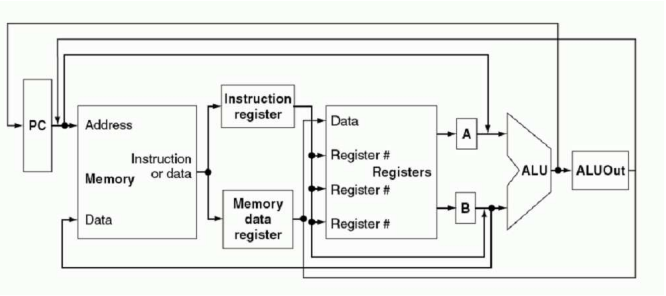
58

微程序设计的多周期MIPS

- Patterson & Hennessy, 附录 D
- 任何 ISA 都可以这样实现

59

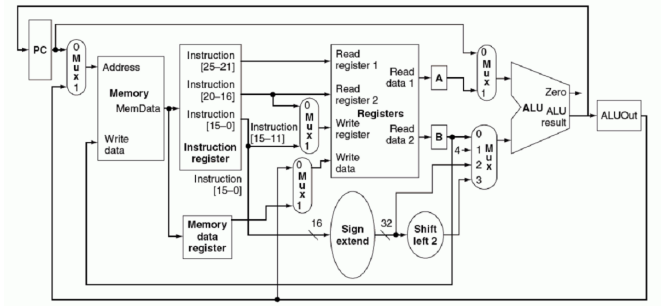
高层抽象的多周期数据通路



[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

60

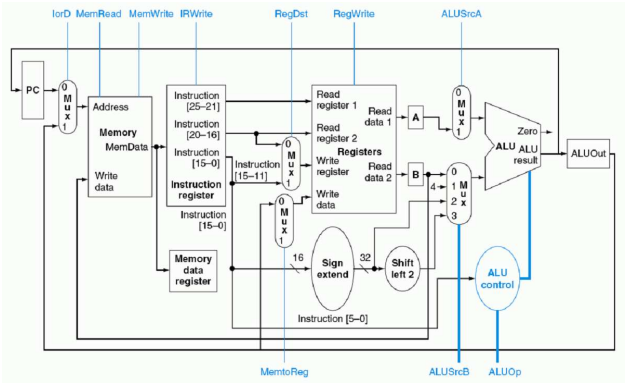
MIPS处理基本指令的多周期数据通路



61

61

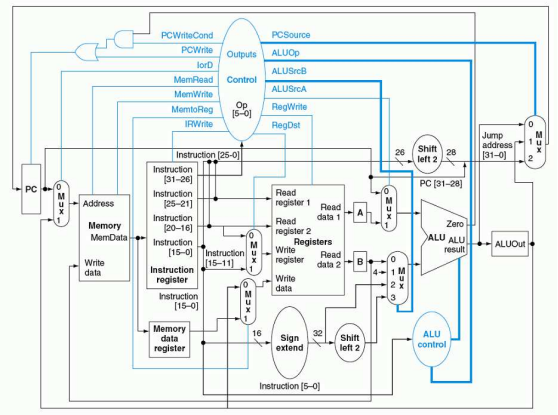
带控制信号的MIPS多周期数据通路



62

62

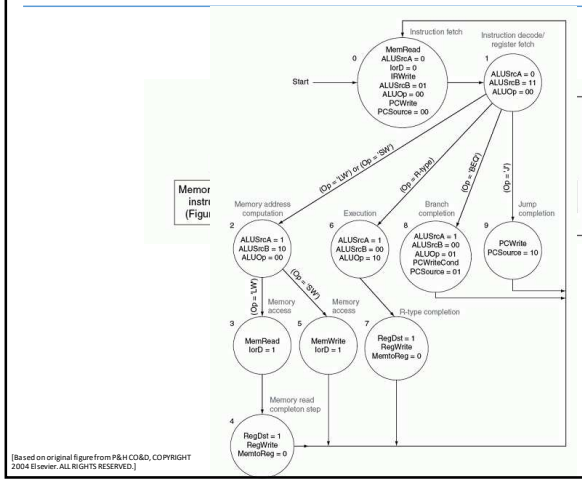
完整的数据通路（带控制信号）



63

63

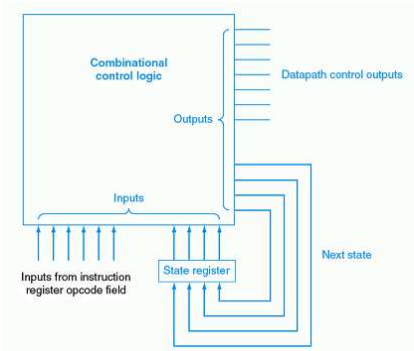
微程序设计的多周期MIPS-状态机



64

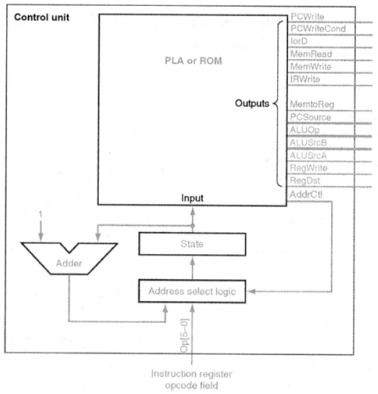
64

MIPS FSM的控制逻辑



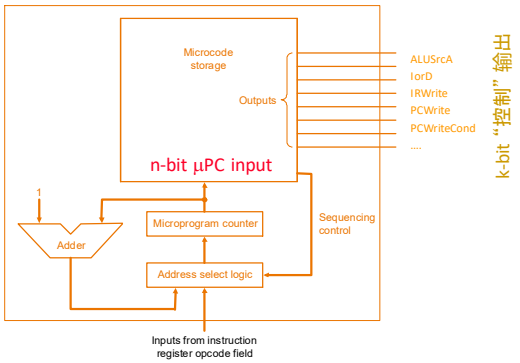
65

MIPS FSM 的微程序设计控制



66

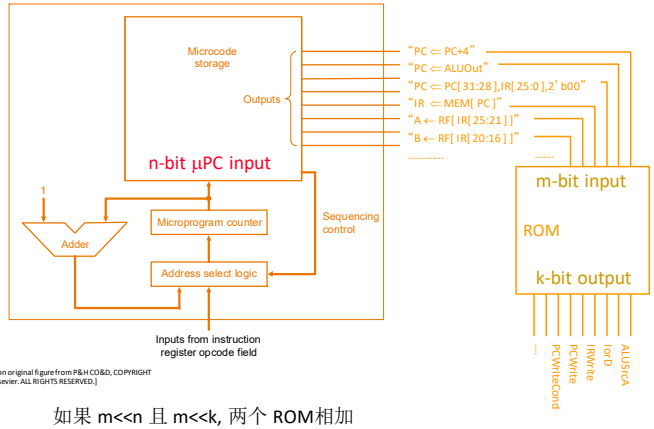
水平微码



控制存储：$2^n \times$ k bit （不包括序列生成逻辑）

67

垂直微码



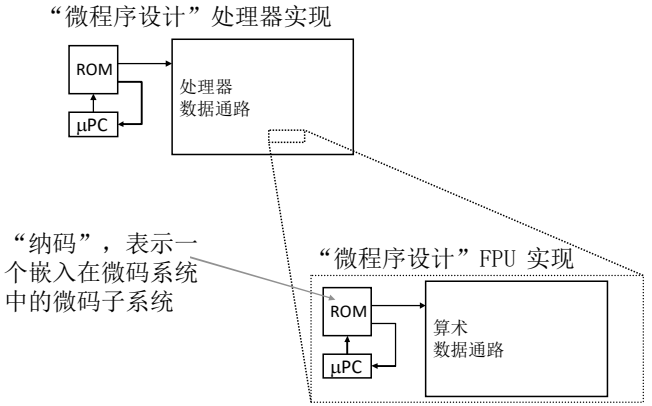
如果 m<<n 且 m<<k, 两个 ROM相加
($2^n \times m + 2^m \times k$ bit) 应该小于水平微码的ROM ($2^n \times k$ bit)

68

“纳码”和“毫码”

- *“纳码”*：比“正牌微码”低一级
  - 为微控制数据通路中的子系统（例如一个复杂的浮点运算模块）做的微程序设计控制
- *“毫码”*：比“正牌微码”高一级
  - 可以被微控制器调用的ISA级别的子程序，用以处理复杂的操作和系统功能
  - 例如，Heller and Farrell，“Millicode in an IBM zSeries processor,” IBM JR&D, May/Jul 2004.
- 在这两种情况下，需要避免将主微控制器复杂化
- 可以理解为不同抽象层次下的“微码”

纳码概念图解



回顾：单周期微架构的复杂性

- 人为因素
  - 所有指令都和最慢的指令一样慢
- 低效
  - 所有指令都和最慢的指令一样慢
  - 必须为所有指令提供最坏情况下的资源
  - 对于一条指令执行周期中在不同阶段会访问同一个资源的情况，必须为该资源提供“副本”
- 不一定是实现ISA的最简单方法
  - REP MOVS, INDEX, POLY等指令的单周期实现?
- 不容易优化/提升性能
  - 对通常情况(普通指令)做优化不起作用
  - 任何时候都要优化最坏的情况

回顾：微体系结构设计原则

- 关键路径设计
  - 找到时延最大的组合逻辑，尽可能的减小它的时延
- 基本（典型）设计
  - 在重要的地方花时间和资源
    - 提升机器设计目标要求的应有能力
  - 通常情况 vs. 特殊情况
- 平衡设计
  - 平衡流过硬件部件的指令/数据流
  - 平衡完成工作所需要的硬件
- *单周期微体系结构是如何遵循这些原则的？*

多周期设计的好处

- 关键路径设计
  - 可以独立地针对每条指令的最糟糕情况来优化关键路径
- 基本（典型）设计
  - 可以通过优化执行“重要”指令（占用大量执行时间）所需的状态数来达到需要的效果
- 平衡设计
  - 不需要提供比实际需求更多的资源或能力
    - 一条指令需要多次使用资源“X”并不意味着需要多个“X”
    - 使硬件更高效：一条指令可以多次***重用***硬件部件

73

73

是否可以更好?

- 在多周期设计中你看到哪些局限?
- 有限的并发
  - 在指令处理周期的不同阶段，一些硬件资源会闲置
  - 例如，当指令在“译码”或“执行”阶段，“取指”逻辑会闲置
  - 当发生访存时绝大多数数据通路闲置

74

74

是否可以利用闲置的硬件改善并发?

- 目标：并发 → 吞吐量（一个周期内完成更多的“工作”）
- 思路：当一条指令在它的处理阶段使用某些资源的同时，使用该指令不需要的闲置资源处理其它指令
  - 例如，当译码一条指令时，取下一条指令
  - 例如，当执行一条指令时，译码另一条指令
  - 例如，当一条指令访问数据存储器时，执行另一条指令
  - 例如，当一条指令写回结果到寄存器堆的时候，另一条指令访问数据存储器

75

75

流水线：基本思想

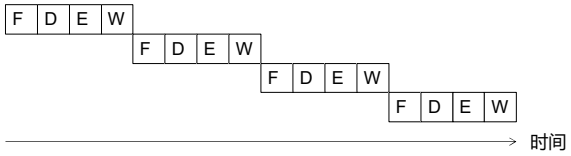
- 系统性更强
  - 多条指令流水线执行
  - 类比：指令的“装配线处理”
- 思路
  - 指令处理周期切分为不同的处理“阶段”
  - 保证有足够的硬件资源在每个阶段处理指令
  - 每个阶段处理不同的指令
    - 指令在连续的阶段中按照程序序连续地处理
- 好处：提升了指令处理的吞吐量（1/CPI）
- 坏处？ 请开始思考这个问题……

76

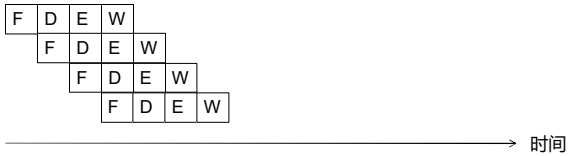
76

例子：执行4条独立的ADD指令

- 多周期：每条指令4个时钟周期



- 流水线：4条指令4个周期（稳定状态）



77

77