计算机体系结构考点梳理

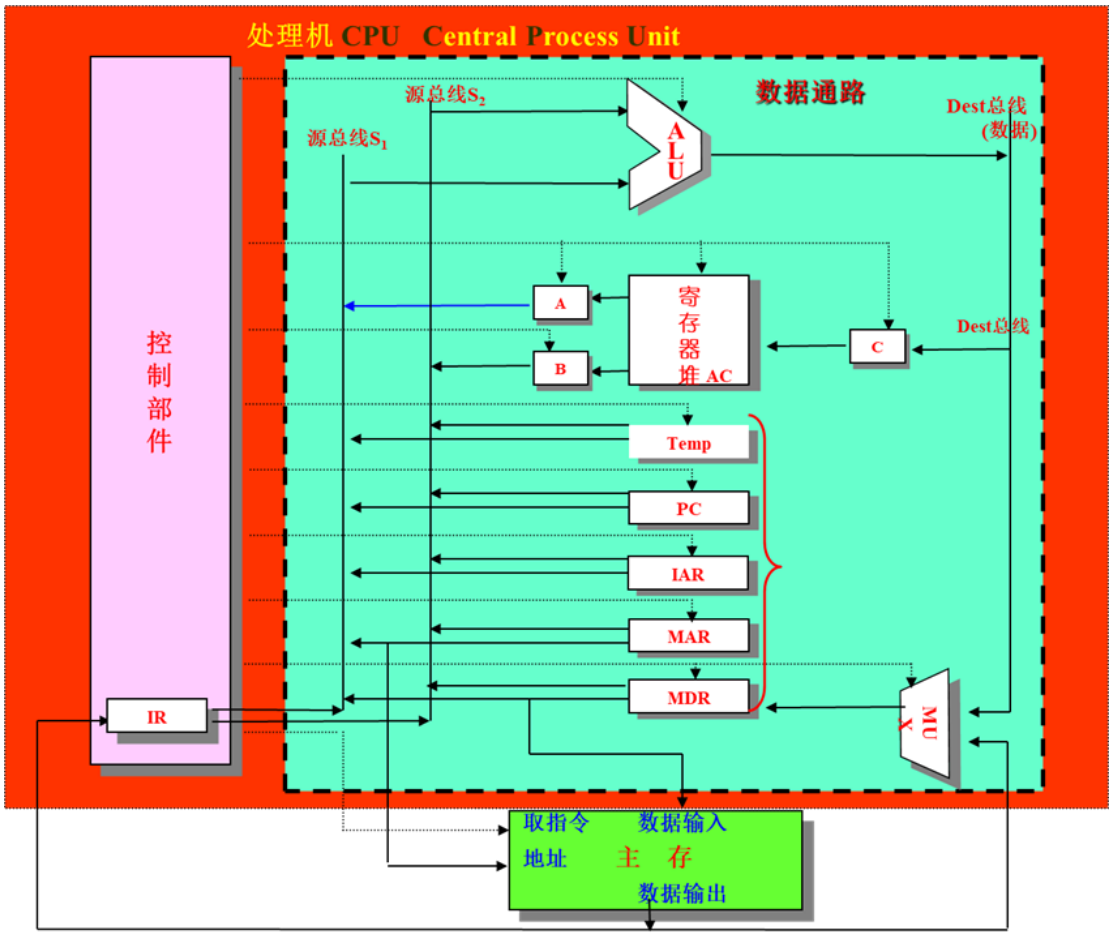
来源：李曦云、李想、刘志林

Hollow Man

目录

老师划定的考点

1. 典型的数据通路组成示意图（大题）



2. 指令执行的五个步骤（填空或大题）

1) 取指令(instruction fetch) 。

2) 指令译码/寄存器读出(instruction decode/register fetch)。

3) 执行/有效地址计算(execution/effective address)。

   a. 访存指令(access instruction)

   b. ALU 指令(ALU instruction)

   c. 无条件转移/条件转移指令(jump instruction / branch)

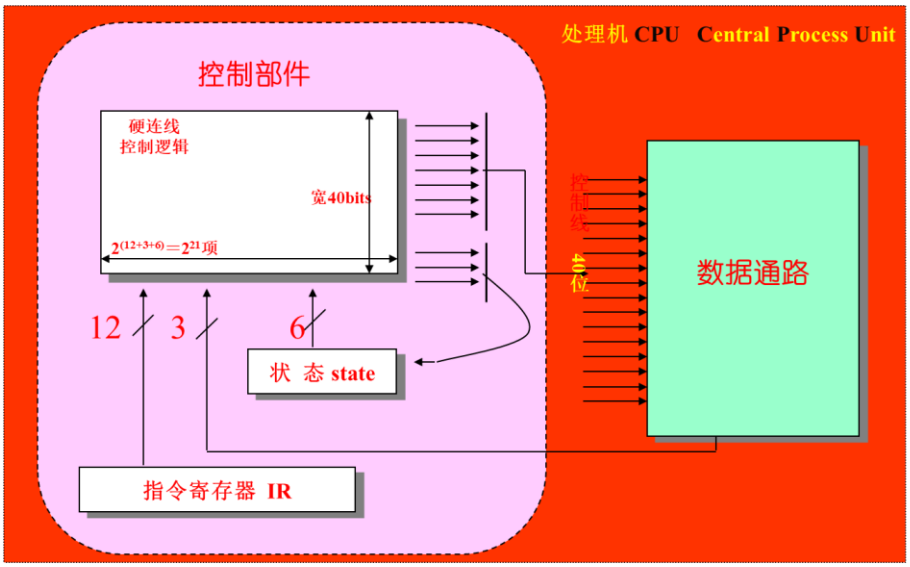
4) 存储器访问/完成转移(memory access/branch completion)。

   a. 访存指令(access instruction)

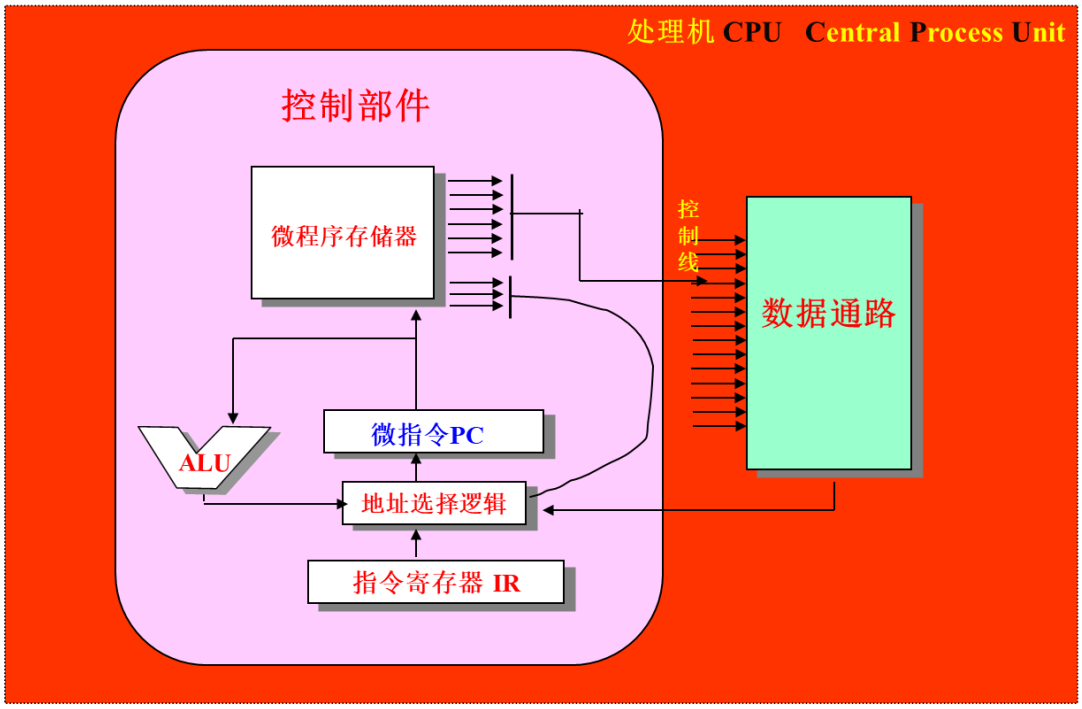
   b. 转移指令(branch instruction)

5) 写回(write-back)。

3. 硬连线控制图（B 卷）



4. 微程序控制图（B 卷）



5. 比较代码指令长度（大题）

	目的寄存器	源寄存器 1	源寄存器 2
0	无	A	B
1	C	Temp	
2	Temp	PC	
3	PC	IAR	
4	IAR	MAR	
5	MAR	MDR	
6	MDR	IR(16 位立即数)	
7		IR(16 位立即数)	
8		常数	

参考例 3-1（ca3.pptx）

设第一张图中机器微指令的三个域源寄存器 1、源寄存器 2、目的寄存器如上图所示，比较垂直微代码和水平微代码的指令长度。

解：由上图可见，目的寄存器有 6 个，加上一种“无目的寄存器”的操作，共需 7 种输出，故水平微代码长度为 7，而垂直微代码长度为[log27]=3。

源寄存器 1 和源寄存器 2 各有 9 种情况，其中 8 种相同，而图 3-1 中，A，B 两个寄存器实际上是相互独立，A 寄存器不可能输出到源总线 S2， B 寄存器也不可能输出到源总线 S1，故而实际上可共用一对信号，于是它们各应有 9 种输出，水平微代码长度为 9，垂直微代码长度[log29]=4，三个域水平微代码总长度 7＋9＋9＝25 字节，垂直微代码总长度为 3＋4＋4＝11 字节，编码使每条微指令节约了 14 字节。

6. 中断名称（填空）

中断在不同的计算机系统中有不同的叫法，典型如下：

- Intel 和 IBM 仍将所有的都称为中断。
- Motorola 将它们称为例外。
- DEC 则根据不同的情况，将它们称作异常、出错、自陷、放弃或中断。

7. 计算机设计者的任务（填空）

计算机设计者的工作是多方面的，包含有：

- 指令集设计；
- 功能组成设计；
- 逻辑电路设计；
- 硬件结构的设计等。

8. 处理器包含的内部部分（填空）

处理器包含数据通路和控制器两部分。

9. 控制存储器（名词解释或判断）

存放微指令的存储器称为控制存储器。

10. 处理器设计与控制器设计的关系（填空或判断）

处理器的设计根本上就是控制器的设计。

11. 计算机需求的功能（名词解释或判断）

计算机需求的功能实际上就是用户需求的功能。

12. 体系结构设计者的目标（填空或判断）

满足计算机的用户需求是体系结构设计者努力追求的目标。

13. 功能需求的实现方式（大题或者填空）

- 直接实现一般指用硬件实现。
- 间接实现是指用软件实现。

14. 设计者和消费者关心的评价标准（填空）

- 设计者最终要用成本性能比作为软、硬件实现功能的取舍标准；
- 消费者要用价格性能比作为选购计算机系统的取舍标准。

15. 设计人员的参考原则（大题）

1) 考虑用户应用领域：对功能单一的专用计算机(控制计算机、专用于传感器模式识别的计算机和算法单一的智能仪器)适当考虑用硬件实现。
2) 设计周期长的硬件不宜采用。
3) 常用的功能尽量采用硬件实现。
4) 实现功能的成本性能比(或价格性能比)要低：既要为生产商接受又要为用户接受。
5) 超前设计：注意计算机及相关技术的发展动态，尽量采用最新的技术和工艺以及超前的设计思想和适当超前的技术和工艺。
6) 遵守法律和道德要求。

16. 软硬件实现的取舍（大题）

参考例 1-1（ca1.pptx）

某一计算机用于商业外贸的事务处理，有大量的字符串处理操作。由于这种商务处理很普遍，有较大的市场，故而设计人员决定在下一代计算机的 CPU 中加入字符串操作的功能。经测试应用软件调查发现，字符串操作的使用占整个程序运行时间的 50%。而增加此功能如用软件(如微程序)实现，则快 5 倍，增加 CPU 成本 1/5 倍；如果用硬件实现，则快 100 倍，CPU 成本增加到 5 倍。问设计人员提出增加此功能是否恰当？如恰当则此功能应该用软件实现还是用硬件实现？设 CPU 成本占整机成本的 1/3。

解：首先来计算机在两种情况下提高的性能和成本性能比。

设： S为 CPU 未增加字符串功能时的 CPU 平均速度，T_{old}为此时运行程序的时间，T_{new}为增加字符串功能后程序运行的时间，则硬件实现有：

$$T_{new}= (1-50\%)T_{old} + \frac{50\%T_{old}}{100}$$

$$性能变化= \frac{T_{old}}{T_{new}} = \frac{T_{old}}{(1-50\%)T_{old} + \frac{50\%T_{old}}{100}} = \frac{1}{(1-50\%)+\frac{50\%}{100}} = 1.98倍$$

$$成本增加= (1-\frac{1}{3}) \times 1+\frac{1}{3} \times 5 = 2.33倍$$

$$成本性能比 = \frac{2.33}{1.98} = 1.18倍$$

软件实现，则有：

$$T_{new}=(1-50\%)T_{old} + \frac{50\%T_{old}}{5}$$

$$性能变化= \frac{T_{old}}{T_{new}} = \frac{T_{old}}{(1-50\%)T_{old}+\frac{50\%T_{old}}{5}} = \frac{1}{(1-50\%)+\frac{50\%}{5}} = 1.66倍$$

$$成本增加= (1-\frac{1}{3}) \times 1+\frac{1}{3} \times (1+\frac{1}{5}) = 1.07倍$$

$$成本性能比 = \frac{1.07}{1.66} = 0.64倍$$

从上面的计算分析看到增加字符串操作功能提高了整机的性能，两种方法均提高性能，且程度相近。但用硬件实现时成本性能比增加了 0.18 倍，而用软件实现时成本性能比却下降了 0.36 倍。

17. 计算机设计的四个原则（填空或大题）

- Amdahl 定律
- 高频事件高速处理(大概率事件优先的原则)
- 局部性原理
- 适应计算机发展趋势

18. Amdahl 定律（名词解释）

某部件应用越频繁，当提高该部件性能时，整机性能也提高的越多。但不管该部件性能提高多大，整机的性能加速不可能大于在原机器中除该部件外所有其它部件运行时间的百分比的倒数 1/(1-F)。

这个定律可以概括为：计算机性能的改善程度受其采用的快速部件(被提高性能的部件)在原任务中使用所占的时间百分比的限制。

19. 计算性能提升（大题）

参考例 1-4（ca1.pptx）

采用新器件使某一功能性能提高 10 倍，但该功能的使用只占原程序运行时间的 40%。请计算新计算机性能改善了多少？

解： $F=0.4$

$S=10$

$$\text{Speedup} = \frac{1}{1-0.4+\frac{0.4}{10}} = 1.56\text{倍}$$

20. 局部性原理（大题或名词解释）

局部性原理，即程序尽可能重复使用它最近使用过的数据和指令，体现在：

- 90/10 局部性规则：程序花费 90%的执行时间运行指令集中 10%的指令代码。
- 时间局部性：如果某一参数被引用，那它不久将再次被引用。
- 空间局部性：如果某一参数被引用，那它附近的参数不久也将被引用。

21. 如何适应计算机发展趋势的因素（大题或者填空）

对一个有经验的设计者来说，其设计的计算机要能经受住：

- 计算机技术发展的考验；
- 硬件技术和软件技术发展的考验；
- 经受住计算机应用市场的冲击。

22. 计算机体系结构设计包含的方面（填空）

计算机系统结构设计主要是计算机的功能设计和指令集的设计。

23. 从属于计算机体系结构设计的分类（大题或者论述或者填空）

计算机系统结构设计主要是计算机的功能设计和指令集的设计。

- 在指令系统中指令的确定是属于计算机系统结构的。
- 指令操作的实现，如取指令、取操作数、运算、送结果等具体操作及排序方式是属于计算机组成的。
- 实现这些指令功能的具体电路、器件的设计及装配技术是计算机物理实现的。

24. CPU 时间包含（填空）

用户 CPU 时间(user CPU time)和系统 CPU 时间(system CPU time)

25. CPI（名词解释）

每条指令的平均时钟周期数(Clock cycles Per Instruction)，简称为 CPI。

26. PCB（名词解释）

为了描述控制进程的运行，系统中存放进程的管理和控制信息的数据结构称为进程控制块（PCB Process Control Block），它是进程实体的一部分，是操作系统中最重要的记录性数据结构。

27. MIPS（名词解释）

million instruction per second，每秒钟执行的百万条指令数。

28. 基准程序的设计原则（大题）

1) 具有代表性，反映用户的实际应用。

2) 不能对基准程序进行优化。

3) 复现性。能重复测试，其环境相同，结果能重复出现。

4) 可移植性。系统相关性要小。

5) 紧凑性。基准程序不宜太庞大。

6) 成本-效率要高。

29. 计算机成本的组成部分（填空）

- 器件成本
- 直接成本
- 间接成本

30. 为什么不能优化基准程序（大题）

现代系统结构要求编译程序根据机器结构对任务代码进行优化，这对提高计算机在实际应用领域中的性能是很好的手段，但基准程序是用来测试计算机性能、测试计算机各种功能实际操作能力强弱，如果对基准程序进行优化，会失去对某些功能的测量，这就失去了测试意义，同时会造成一种高性能的假象。

31. 计算条件转移成功率（大题）

假定反向条件转移指令 90%是成功的，用下表平均数计算正向条件转移的成功率。

基准程序	反向条件转移	条件转移成功	全部条件指令
Gcc	26%	54%	63%
Spice	31%	51%	63%
Tex	17%	54%	70%
平均	25%	53%	65%

解：平均转移指令的成功率包含正向与反向两部分成功的转移指令，假设成功的反向条件转移指令为B_{taken}，成功的正向条件转移指令为F_{taken}，设条件转移指令共有SUM条。则

$$\mathbf{Sum\times B_{taken}\times 25\%+Sum\times F_{taken}\times (1\text{-}25\%)=Sum\times 53\%}$$
$$\mathbf{90\%\times 25\%+F_{taken}\times 75\%=53\%}$$
$$\mathbf{F_{taken}=(53\%\text{-}22.5\%)/75\%=40.7\%}$$

从例题中我们可以发现一个很有用的结论：正向条件转移大部分是不成功的，它满足条件的概率较低。

32. 主存储器编制方法（填空或名词解释或大题或者判断等）

- 低位收尾(little endian)，其字节次序是低字节在最低位的排列（DEC/Intel）
- 高位收尾(big endian)，其字节次序是高字节在最低位的排列（IBM/摩托罗拉）

33. 对齐电路的分类（填空或者名词解释）

- 外部对齐是指 CPU 和外部器件交换时的数据对齐关系。
- 内部对齐是 CPU 内部数据总线和寄存器之间的数据对齐关系。

34. 编址方式（填空或者大题）

- 统一编址是把除外存以外的各种存储部件的地址统一编成二进制码。如主存、通用寄存器、各种 I/O 接口寄存器等。
- 单独编址又称局部编址，是对各种不同的存储空间分别编址。如主存、通用寄存器、I/O 接口寄存器等。
- 隐含编址多用于一些专用的存储单元。如指令计数器、程序状态字、堆栈、专用累加器等。

35. 使用概率最高的指令（填空）

ALU、传送和转移指令。

36. RISC 思想精髓或设计原则（大题）

“Simple is fast”和“Small is fast”，即：简单事件可以更快速处理；小规模器件的速度可以做的更快。

37. RISC 设计思想（大题）

指令集设计时根据 Amdahl 定律选择使用概率高的指令构成指令集，这些大概率指令一般是简单指令，因此控制器可以设计的简单、高速，且占 CPU 集成电路芯片的面积少，空出较多的集成电路芯片面积用来增加寄存器数量。在编译的配合下减少访存次数，减少指令间的各种相关和竞争，尽可能得到最佳指令序列，从而提高计算机系统的整体性能。

38. 流水线的作用（大题）

提高硬件功能部件的使用率，减少指令的平均执行时间。

39. 流水线技术的概念（大题或者名词解释）

流水线(pipeline)是指在程序执行时多条指令重叠进行的操作的一种准并行处理实现技术。

40. 流水线的任务（大题或者填空）

确定每个流水级功能部件处理事件的时间，平衡各流水级处理部件的处理时间。

41. 流水线深度（名词解释）

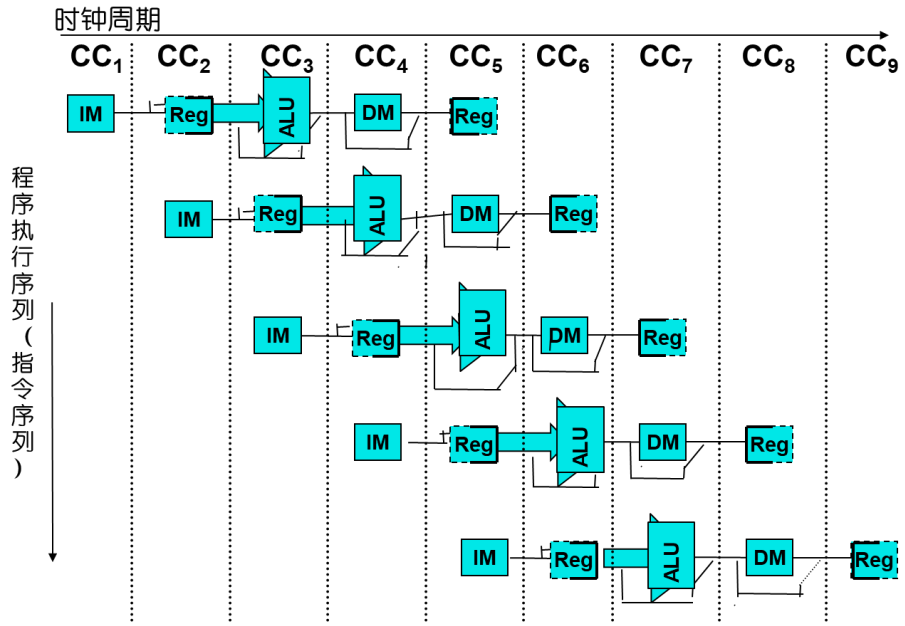
流水线深度是指流水线中总的流水级的数目。

理想条件下流水线计算机的加速比(speedup)就等于流水级的深度。

42. 流水线状态图（大题）

指令序列	流水时钟数								
	1	2	3	4	5	6	7	8	9
指令$_i$	IF	ID	EX	MEM	WB				
指令$_{i+1}$		IF	ID	EX	MEM	WB			
指令$_{i+2}$			IF	ID	EX	MEM	WB		
指令$_{i+3}$				IF	ID	EX	MEM	WB	
指令$_{i+4}$					IF	ID	EX	MEM	WB

43. 流水线功能部件图（大题）



44. 流水线基本结构（填空）

- 处理的级别：操作部件级、指令集流水、处理器级流水。
- 完成的功能：单功能流水线、多功能流水线
- 连接方式：静态流水线、动态流水线
- 处理的数据：标量流水线、向量流水线
- 流水线结构：线性流水线、非线性流水线

45. 流水线工作方式（填空或者判断）

准并行。

46. 流水线的竞争种类（大题）

- 结构竞争(structure hazard)(资源竞争)：由资源缺乏引起。
- 数据竞争(data hazard)：由指令间数据相关而引起。
- 控制竞争(control hazard)：由程序指针 PC 值的改变而引起。

47. 流水线的竞争概念（名词解释或者大题）

在流水线中工作的指令流，由于某种原因，阻碍了指令流中的下一条指令在预定的时钟周期内作相应流水级的操作的现象。我们称这种现象为流水线竞争。

48. 设计者允许结构竞争的原因（大题）

- 减少成本。
- 降低单元电路的延时时间。
- 减少电路的复杂程度。

49. 数据竞争的种类（大题）

- 先写后读相关 RAW (read after write) （写读冲突）
- 写写相关 WAW (write after write) （写写冲突）
- 先读后写相关 WAR (write after read) （读写冲突）

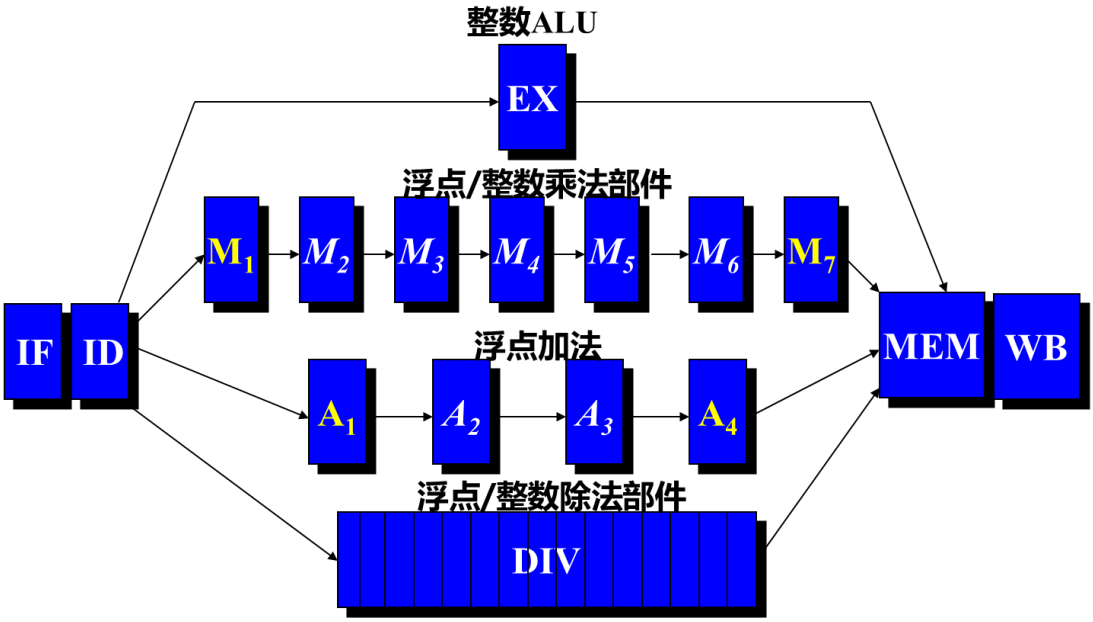
50. 指令发射的概念（名词解释）

指令从 ID 级流入 EX 级，一般称指令发射(instruction issue)。一条指令已建立了这一过程，称为已发射(issued)。

51. 整数与浮点的平均预测出错率（判断）

- 浮点程序的平均预测出错率是 9%，偏差是 4%；
- 整数程序的预测出错率是 15%，偏差是 5%。

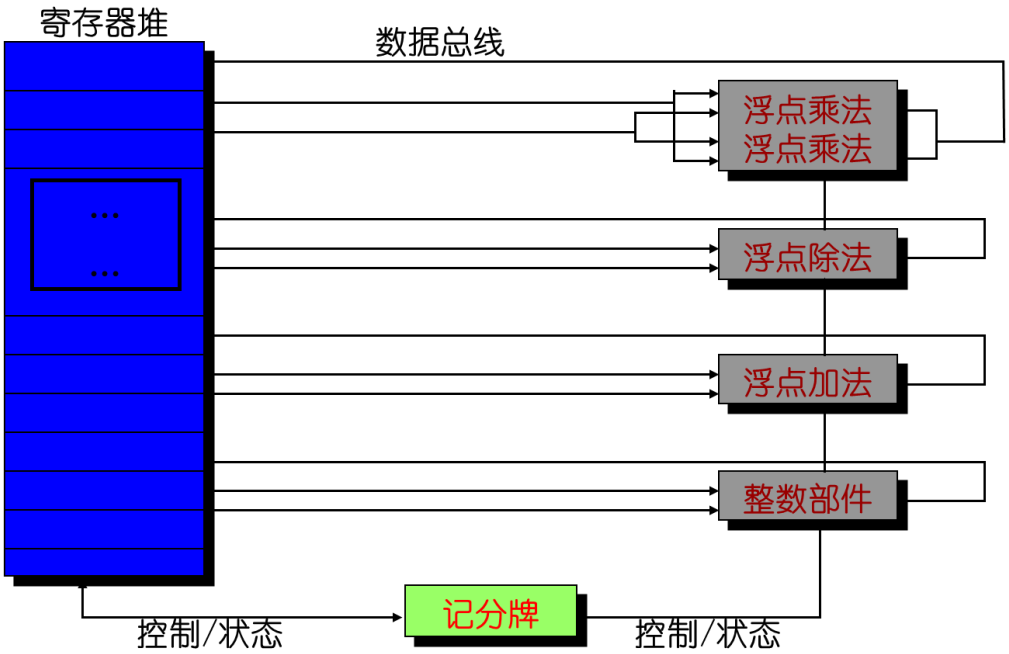
52. 基本流水线扩展图（大题）



53. 记分牌的概念（大题）

记分牌动态调度算法也称为集中式动态调度，在动态流水线调度中，所有指令有序地通过发射级。但进入读操作级后，每条指令有的立即处理，有的暂停处理，使后面几级流水线指令的执行变成乱序进行。记分牌(score boarding)就是一种允许指令乱序执行，克服资源和数据冲突的技术。

54. 记分牌的基本结构（大题）



55. 记分牌的四级流水步骤（大题）

1) 发射级(issue) //*处理结构竞争和 WAW 竞争
2) 读操作数(read operands) //*动态解决 RAW 竞争
3) 执行(execution)
4) 写结果(write result) //*处理 WAR 竞争

56. 记录记分牌结果的三张表格（大题）

要求会画，需要推导，以下图仅供参考。

指令状态表：

指令	指 令 状 态①			
	发射 issue	读操作数 read operands	执行 execution	写结果 write result
LD F_6, 34(R_2)	√	√	√	√
LD F_2, 45(R_3)	√	√	√	
MULTD F_0, F_2, F_4	√			
SUBD F_8, F_6, F_2	√			
DIVD F_{10}, F_0, F_6	√			
ADDD F_6, F_0, F_2				

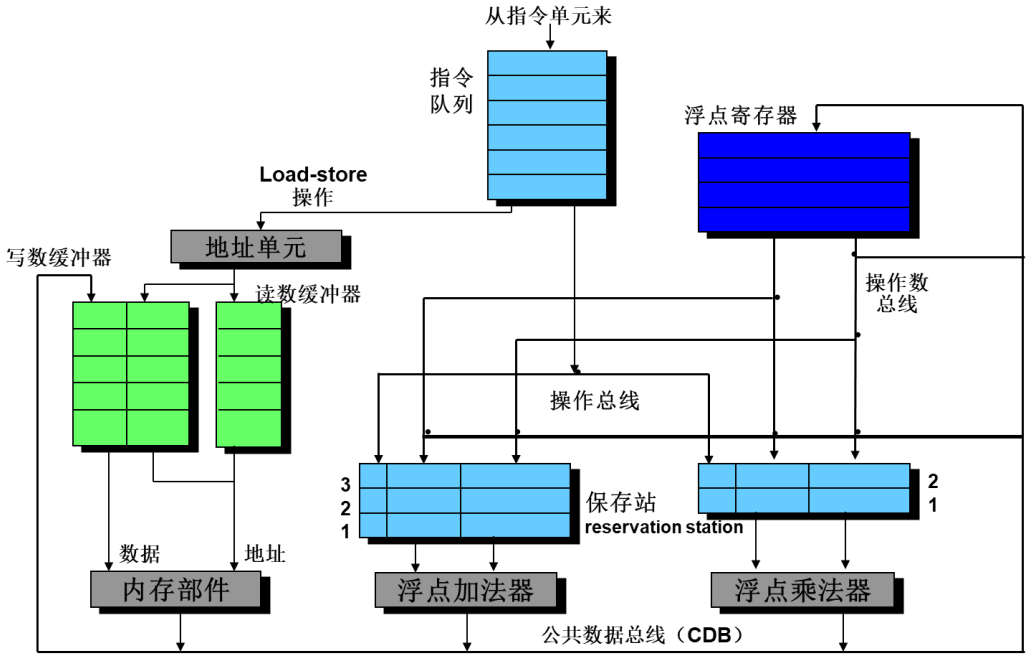
功能部件工作表：

部件名称	功 能 部 件 工 作 状 态②								
	Busy	Op	F_i	F_j	F_k	Q_j	Q_k	R_j	R_k
Integer	Yes	Load	F_2	R_3				No	
$Mult_1$	Yes	Mult	F_0	F_2		Integer		No	
$Mult_2$	No								
Add	Yes	Sub	F_8		F_2		Integer		No
Divide	Yes	Div	F_{10}	F_0		$Mult_1$		No	

寄存器结果状态表：

域	寄 存 器 结 果 状 态③								
	F_0	F_2	F_4	F_6	F_8	F_{10}	F_{12}	…	F_{30}
FU	$Mult_1$	Integer			Add	Divide		…	

57. Tomasulo 算法浮点部件基本结构（B 卷）



58. Tomasulo 算法 3 级处理（B 卷）

- 发射级(Issue，记为 IS)。
- 执行级(Execute，记为 EX)。
- 写结果级(Write result，记为 WR)。

59. Tomasulo 算法调度（B 卷）

三张表格（略）

60. 多发射处理器类型（填空或判断）

- 超标量处理器(superscalar processor，SP)，静态+动态；
- 超长指令字(very long instruction word，VLIW)处理器，静态。

61. 路径调度的两种独立过程（填空）

- 路径选择
- 路径压缩

62. 评价存储器的参数（大题）

⑴ 容量：$S=W\times l\times m$ 表示。其中W为存储器字长，l为存储器字节，m为存储器个数。

⑵ 速度：

访问时间(access time)T_a：从存储器接到读请求到所读的字传送到数据总线上的时间间隔。

存储周期T_m：连续两次访问存储器之间所必需的最小时间间隔。一般，$T_m > T_a$。

存储带宽B_m：存储器被连续访问时所提供的数据传输速率，单位是位(或字节)／s。

⑶ 价格：通常用单位字节价格来表示。若总容量为S的存储器的总价格为C，则单位字节价格 $c = C／S$。

63. 存储体系设计目标（填空）

高速度、大容量、低价格。

64. 块（名词解释）

相邻两级存储器之间信息交换的最小单位。块大小一般是固定的，但也可以是可变的。若块的大小固定，则两级存储器的容量为块大小的整数百倍。

65. 命中率（名词解释）

CPU 产生的有效地址可以在高层存储器中访问到的概率。

66. 失配率（名词解释）

CPU 产生的有效地址不能直接在高层存储器中访问到的概率。

67. 命中时间（名词解释）

访问到高层存储器所需的时间，其中包括本次访问是命中还是失配的判定时间。

68. 平均存储访问时间（名词解释）

average memory_access time,AMT。平均存储访问时间＝命中时间＋失配率×失配时间

表示为：AMT＝HT＋M×MP

69. Cache 写策略（大题）

- 直写（write through）：信息被写入 Cache 行的同时，利用 CPU 和主存之间的直接数据通路写入主存的对应块中。
- 回写（write back）：信息只写入 Cache 的相应行，仅当被修改过的行被替换出 Cache 时，才将它写入主存的相应存储块中。

70. Cache 读命中的工作流程（大题）

1) 来自 CPU 的地址被分为 29 位块帧地址和 3 位块内偏移地址，块帧地址又分成 20 位标志和 9 位索引。
2) 根据索引选择 Cache 中的一个组，读取组内各行标志以判定要访问的数据块是否在 Cache 中。
3) 块帧地址的标志域与步骤 2 中读取的两个行标志作相等比较。
4) 假设有一行标志与块帧地址的标志相匹配，则由 2 选 1 多路转换器选取相应的数据行。
5) 读出的字送往 CPU。

71. 伪关联 Cache 命中方式（大题）

- 快的命中: 对应的命中时间即常规的命中时间。
- 慢的命中: 对应的伪命中时间（一般伪命中时间>常规命中时间）。

72. 虚拟存储器分类（填空）

- 块大小固定不变的页式虚拟存储器
- 块大小可变的段式虚拟存储器

73. WSC（名词解释）

Worehouse Scale Computer，仓库级计算机，是商用因特网基础的超大规模集群，许多人每日所用 Internet 服务的基础。

74. 指令按照应用范围可以分为三类：

1.基本指令：供用户使用的最常用指令。

2.专用指令：应用领域的特殊设计，也可供用户使用。

3.特权指令：仅供系统程序员使用，不允许普通用户使用。

75.设计计算机的功能考虑：

并不就是用户所需要的功能，它主要考虑成本、技术、兼容性、市场大小等诸多因素。

76. 判断：

指令功能性越强，移植性越差。

完成同样任务所需时间越短，计算机性能越好。

如果用响应时间评价性能，我们称短为性能好。

超级流水线可发射多个过程。

基本流水线可发射两个过程。（错）

77. 2-14：

RISC设计技术

[例2-14] 某应用程序，简单的基本指令占80%，而复杂指令占20%，在传统CISC计算机上运行，简单指令的CPI=4，复杂指令的CPI=8，而在RISC计算机上运行只有简单指令，其CPI=1，复杂指令用简单指令合成实现。假设平均每条复杂指令需14条基本指令组合，请比较两个计算机系统运行该应用程序的性能。

解：设CISC计算机的时钟周期为 T_{CISC}，RISC计算机的时钟周期为T_{RISC}，则：

CISC计算机的平均

$$CPI=0.8\times4+0.2\times8=4.8$$

4.2 RISC设计的原则

RISC设计技术

RISC计算机的指令数增加为

$$I_{RISC}=0.8I_{CISC}+0.2I_{CISC}\times14=3.6I_{CISC}$$

因 $CPU_{time}=I\times CPI\times T$

故 $CPU_{time\text{-}CISC}=I_{CISC}\times4.8\times T_{CISC}=4.8I_{CISC}\times T_{CISC}$

$$CPU_{time\text{-}RISC}=I_{RISC}\times1\times T_{RISC}=3.6I_{CISC}\times T_{RISC}$$

计算机的性能比 = $\frac{4.8\,I_{CISC}*T_{CISC}}{3.6\,I_{CISC}*T_{RISC}}=1.33\,\frac{T_{CISC}}{T_{RISC}}$

若$T_{CISC}=T_{RISC}$，则RISC计算机比CISC计算机性能提高33%。

实际上，减少T_{RISC}比减少T_{CISC}要来的容易。

78. I/O 设备分类：

数据表示设备、网络通讯设备、存储设备

1.1 并行体系结构的分类

用多处理器来增加性能、提高可靠性的想法，可以追溯到最早的计算机。

弗林(Flynn) [1966年]提出了一个计算机分类的简单模型，他根据机器最关键部位的指令和由指令引起数据流的并行性，把所有的计算机分为四类：

1. 单指令流，单数据流(SISD)——这就是一个单处理器。
2. 单指令流，多数据流(SIMD)——同一指令由多个处理器执行，这些处理器使用不同数据流，有各自的数据内存，但共享一个指令内存和控制处理器（负责存取和发送指令）。处理器通常是专用的，不要求通用性（多媒体扩展就是SIMD并行的一种形式，向量系统结构是这种系统结构的最大子类）。
3. 多指令流，单数据流(MISD)——多个功能单元仅对单个数据流操作（一些专用的流式处理器接近这种形式）。
4. 多指令流，多数据流(MIMD)——每个处理器存取自己的指令，操作自己的数据。处理器通常就采用普通的微处理器。

1.1 并行体系结构的分类

现有的MIMD机器基于使用的处理器数目，可以分为两类，并由此决定了存储器组织和互联方案。我们将根据存储器组织来命名这些机器，因为处理器数目多少的量度是会随时间改变的。

第一种类型称为“对称(共享存储器)多处理器”(symmetric multiprocessors,SMP)。或“集中式共享存储器多处理器”(centralized shared-memory architecture)。在90年代，这种类型使用几十个处理器，处理器间通过总线互联共享一个集中式存储器，运用大量高速缓存，总线和单个存储器可以满足小数目处理器对内存的需求。由于只有一个主存储器，不同的处理器对它的存取时间相同，因此，这些机器有时被称作统一存储器访问(Uniform Memory Access,UMA)型。这种结构是目前最为流行的组织方式。图8-1给出了这种机器的形式：

注：《计算机科学》期刊的论文模板

论文题目*)

作者

(大学 学院 长沙 410073)

摘 要 内容 的研究,针对 ，利用 给出 ， 。

关键词 理论,逻辑

- **Paper title**

Author

(School of , University , Changsha, 410073, China)

Abstract study on.

Keywords Ttheory, Logic

1 引言

在计算机科学领域,

2 类型理论

类型理论最初被设计用来作为形式化构造性数学的基础[9]，但近年来，科学家们发现了很多它在计算机科学方面的应用。

2.1 命题

命题实质上就是描述性质和事实的公式，

2.2 逻辑框架(LF)

逻辑框架可以以多种不同的方式被应用。

3 标题

传名调用和传值调用方法可以借用下面的替换加以描述。

应用的计值规则可以如图 2 那样形式地描述。

从图 2 我们可以清楚地看出，

图2 传名与传值

图2 传名与传值

定义 4 类型。

定义 5

引理 1

定理 1

结语

通过前面对。

参考文献

[6] Harper R, Honsell F, Plotkin G. A Framework for Defining Logics[C]. Proc 2nd Ann Symp on Logic in Computer Science IEEE，1987

[7] 庞建民，赵荣彩.Haskell 语言的列表内涵特性及其应用.计算机工程与应用[J]，2005，41(4):99-101

PPT 内容整理

第零章 引入

1. 计算机体系结构研究内容

- 从外部来研究计算机系统；

- 使用者所看到的物理计算机的抽象；
- 编写出能够在机器上正确运行的程序所必须了解到的计算机的属性；
- 软硬件功能分配及分界面的确定。

2. 系统

为完成特定任务由相关部件或要素组成的有机的整体。

3. 计算机系统的组成方式

- 由运算器、控制器、存储器、输入和输出 5 个部件组成。
- 由紧密相关的硬件和软件组成。
- 由人员、数据、设备、程序、规程 5 部分组成。

4. 阿塔纳索夫提出的计算机的三条原则

- 采用二进制运算，以保证精度；
- 采用电子技术来实现控制和运算，以保证计算速度；
- 采用计算（运算和控制）功能和存储功能相分离的结构。

5. 摩尔定律

也称为计算机第一定律，当价格不变时，集成电路上可容纳的元器件的数目，约每隔 18-24 个月便会增加一倍，性能也将提升一倍。

6. 计算机的分类

- 个人移动设备 (PMD)
- 桌面计算（Desktop Computing）
- 服务器（Servers）
- 集群/仓库级计算机（Clusters / Warehouse Scale Computers）
- 嵌入式计算机（Embedded Computers）

7. 软件人员应具备的素质

- 团体软件开发；
- 智商 IQ、情商 EQ；
- 素质、思维(创意)，包括社交能力、人性化、热情、自我管理等。

第一章 计算机设计基础

1. 计算机设计者的几大任务

- 指令集设计
- 功能组成设计
- 逻辑电路设计
- 硬件结构的设计等

2. 计算机需求的功能

计算机需求的功能实际上就是用户需求的功能。

3. 体系结构设计者努力追求的目标是什么

满足计算机的用户需求是体系结构设计者努力追求的目标。

4. 筛选功能需求的依据

筛选的依据是成本性能比。

5. 功能需求的实现方式

- 直接实现一般指用硬件实现
- 间接实现是指用软件实现

区分的界限就是成本性能比和用户的承受能力。

6. 软件实现的优缺点

- 优点：低成本、易设计、低出错率、可改性强、适用性强、设计周期短以及升级提高较简单。
- 缺点：执行速度慢，一般以牺牲时间来实现其功能的；另外，相对的存储器费用和软件设计的费用也要增加。

7. 硬件实现的优缺点

- 优点：提高计算机在这一方面的性能（速度）
- 缺点：造价昂贵，设计周期很长，市场适应能力差。

8. 取舍硬件软件实现的原则

常用的基本功能或产量很大的功能才适宜于用硬件实现。

软件和硬件要综合考虑，要以价格性能比高低为取舍原则。

9. 设计参考原则

1) 考虑用户应用领域：对功能单一的专用计算机适当考虑用硬件实现。
2) 设计周期长的硬件不宜采用。
3) 常用的功能尽量采用硬件实现。
4) 实现功能的成本性能比(或价格性能比)要低：既要为生产商接受又要为用户接受。
5) 超前设计：注意计算机及相关技术的发展动态，尽量采用最新的技术和工艺以及超前的设计思想和适当超前的技术和工艺。
6) 遵守法律和道德要求。

10. 消费者和设计者的取舍标准

- 设计者最终要用成本性能比作为软、硬件实现功能的取舍标准
- 消费者要用价格性能比作为选购计算机系统的取舍标准

11. 计算机设计的几个原则

- Amdahl 定律
- 高频事件高速处理(大概率事件优先的原则)
- 局部性原理
- 适应计算机发展趋势

12. Amdahl 定律

某部件应用越频繁，当提高该部件性能时，整机性能也提高的越多。但不管该部件性能提高多大，整机的性能加速不可能大于在原机器中除该部件外所有其它部件运行时间的百分比的倒数 1/(1-F)。

可以概括为计算机性能的改善程度受其采用的快速部件(被提高性能的部件)在原任务中使用所占的时间百分比的限制。

13. 高频事件高速处理(大概率事件优先的原则)

如果某一功能使用越频繁，那么它对整机性能的影响也就越大。故高频事件高速处理，即为常用器件做的快一些。

14. 局部性原理

程序尽可能重复使用它最近使用过的数据和指令。

- 90/10 局部性规则：程序花费 90%的执行时间运行指令集中 10%的指令代码。
- 时间局部性：如果某一参数被引用，那它不久将再次被引用。
- 空间局部性：如果某一参数被引用，那它附近的参数不久也将被引用。

15. 适应计算机发展趋势

对一个有经验的设计者来说，其设计的计算机要能经受住计算机技术发展

的考验、硬件技术和软件技术发展的考验、更要经受住计算机应用市场的冲击。因此设计者必须知道计算机技术和计算机应用的发展趋势。在设计时要有超前意识，这样才能增强计算机的生命力。

16. 计算机系统设计的主要方法

- 自下而上设计：不管应用要求，只根据能拿到的器件参照或吸收已有各种机器的特点，先设计出微程序机器级（如果采用微程序控制）及传统机器级，然后再为不同应用配多种操作系统和编译系统软件。
- 自上而下设计：先考虑如何满足应用要求，确定好面对使用者那级机器应有什么基本功能和特性，如基本命令、指令或语言结构、数据类型和格式等，然后再逐级往下设计，每级都考虑怎样优化上一级实现。
- 由中间开始设计：先进行合理的软、硬件功能分配，既要考虑能拿到的硬（器）件，又要考虑可能的应用所需的算法和数据结构，先定义好这个交界面。确定哪些功能由硬件实现，哪些功能由软件实现，同时还要考虑好硬件对操作系统、编译系统的实现提供些什么支持。然后由这个中间点分别往上、往下进行软件和硬件的设计。

17. 如何设计好的计算机体系结构

要设计一个好的计算机体系结构还必须与应用领域(处理对象)结合起来，在问题(处理对象)、算法、数据结构与硬件结构之间找到一个最佳映射。

18. 计算机组成设计

计算机组成设计是指计算机系统结构的逻辑实现。主要是指机器级功能的硬件逻辑设计，包括机器级的数据流和控制流的组成设计及其逻辑功能设计。

19. 计算机设计中各类别的实现

- 在指令系统中指令的确定是属于计算机系统结构。
- 指令操作的实现，如取指令、取操作数、运算、送结果等具体操作及排序

方式是属于计算机组成。

- 实现这些指令功能的具体电路、器件的设计及装配技术是计算机物理实现。

20. 衡量计算机性能的参数

响应(实耗)时间是指计算机系统完成某一任务(程序)所花费的时间，是衡量计算机性能的重要依据。

21. 响应时间

响应(实耗)时间是指计算机系统完成某一任务(程序)所花费的时间。它包含磁盘访问、存储器访问、输入/输出等待所需要的时间以及操作系统所开销的时间。在多道作业系统中还包括各任务之间的切换和等待。

22. 计算机速度

是指每秒钟完成的事件数目，即用响应时间的倒数。

23. 计算机整机性能的组成部分

- CPU 执行程序的时间
- 等待时间

提高计算机性能就是要提高 CPU 性能和减少等待时间

24. CPU 时间的组成部分

- 用户 CPU 时间
- 系统 CPU 时间

25. CPI

每条指令的平均时钟周期数。

26. 影响 CPU 性能的因素

- 时钟频率（f）：由硬件技术和组成技术决定。
- CPI：与组成及指令集有关。
- 指令数（I）：与指令集和编译技术有关。

27. MIPS

每秒钟执行的百万条指令数。

28. MIPS 的注意点

1) MIPS 的大小和指令集有关，不同指令集的计算机之间的 MIPS 不能比较。
2) 同一台计算机上的 MIPS 是变化的。
3) 有时 MIPS 会出现矛盾。

29. MFLOPS

每秒钟执行的百万个浮点操作数。

30. 工作负载基准程序

计算机专家们从实际的应用中归纳一系列测试程序，用来测试机器的软件和硬件的综合性能，我们称这些测试程序为基准程序。

31. 工作负载基准程序的组成

- 实际程序
- 核心基准程序
- 简单基准程序
- 合成基准程序

32. 构成基准程序的方法

- 实际工作负载中抽取代码称为自然测试负载
- 为了测试而专门构成的程序代码称之为人工测试负载

33. 基准程序的准则

1) 能代表被测试系统的工作负载，包括系统的基本语言和主要应用方面
2) 检查各种功能，例如作业调度、文件管理、I/O 支持以及语言处理等

34. 基准程序的层次

- 物理层的基准程序是面向物理资源
- 逻辑层是面向资源
- 应用层是面向用户应用

35. 基准程序的设计原则

1) 具有代表性，反映用户的实际应用。
2) 不能对基准程序进行优化。
3) 复现性。能重复测试，其环境相同，结果能重复出现。
4) 可移植性。系统相关性要小。
5) 紧凑性。基准程序不宜太庞大。
6) 成本-效率要高。

36. 性能报告的常用方法

1) 总执行时间统计法
2) 加权执行时间的统计法
3) 几何平均
4) 规范化总执行时间

37. SPEC 基准程序

在 1988 年，美国 HP、DEC、MIPS 以及 SUN 公司，发起成立了 SPEC 组织(系统性能评价合作团体)，用一组实用程序和相应输入来评价计算机系统性能。

38. 计算机成本的组合

- 器件成本：物理硬件成本
- 直接成本：直接花在产品上的费用，它包括工时、元器件报废成本，以及保修等一切与计算机生产直接有关的费用，一般是器件成本的 25%~40%。
- 间接成本：或称毛利润(gross margin)是生产厂商在生产过程中花费的，但不能直接计算到产品上的费用，如厂房等固定资产费用、研制开发费用(R&D)、市场推销费用、设备折旧维护费、税前利润及税费。这部分一般是平均售价的 45%~65%。

39. 平均销售价

平均销售价是每台计算机产品销售后要直接返回给计算机生产厂商的，相当于订价。

40. 报价单价格和平均销售价的区别

用户直接接触到的是产品的报价单价格，而不是平均销售价。报价单价格还要在平均销售价上加上一个差价，这部分差价是用于零售商的利润和用户的折扣，它们占保价单价格的 25%~40%。

41. 用体系结构知识选购计算机

1) 多余的功能是浪费。
2) 要针对应用范围选择测试机器。
3) 要考虑整机的工作性能。

4) 考虑计算机的发展趋势。

5) 价格性能比和承受能力。

第二章 指令集的设计

1. 指令的作用

指令的作用是协调各硬件部件之间的工作关系，它反映了计算机所拥有的基本功能。

2. 描述指令集的结构特点的几个方面

1) 操作数在 CPU 内的存储形式。操作数除了存放在存储器中之外，还可存放在 CPU 内的什么地方。（最基本特点）

2) 每条指令中显示说明的操作数个数。

3) 操作数的位置。 ALU 指令的操作数可以放在存储器[指明寻址方法]中，也可存放在 CPU 内部寄存器中。

4) 操作类型。 指令中直接提供的操作功能有哪些，即不同功能的指令种类。

5) 操作数的类型和长度。每个操作数的类型和长度是什么，如何说明类型和长度。

3. 三种机器结构

- 堆栈结构（隐含在栈顶，无显式操作数）

  优点：赋值表达式简单，指令长度较短，代码密度高；

  缺点：不能随机访问存储器，代码效率低。

- 累加器结构（隐含在累加器中，有 1 个显式操作数）

  优点：机器内部状态最少，指令长度最短（9bytes）；

  缺点：仅一个暂存器，和存储器的通信频繁。

- 通用寄存器结构（显式书名在寄存器或存储器中，有 2-3 个显式操作数）

  优点：最一般的指令模型，第一类，寄存器利用率最高，代码长度 MAX＜15 字节，第二类，和累加器形式相似，暂存器个数多；

缺点：寄存器要显示说明，导致指令字较长。

4. 三种指令类型

- 在 ALU 指令中不对内存进行操作的计算机称为载入-存储(Load-Store)或者寄存器-寄存器(register-register)指令。
- ALU 指令中有一个内存操作数的指令称为寄存器-存储器(register- memory)指令。
- 有多个内存操作数的指令称为存储器-存储器(memory-memory)指令。

5. 指令的应用范围分类

- 基本指令：供用户使用的最常用指令。
- 专用指令：应用领域的特殊要求设计，也可供用户使用。
- 特权指令：仅供系统程序员使用，不允许普通用户使用。

6. 最常用指令

- 控制操作（最重要）
- 算术操作（最多）
- 数据传输

7. 指令集设计的方法和原则

四大方法和原则：

1) 应用范围；
2) 指令的使用概率；
3) 常用指令分析；
4) 特殊指令设计。

设计时要考虑的方面：

1) 正交性：指令的各个编码段之间应是不相关的，如代码段、数据段、寻址等的编码应彼此独立，互不相关。

2) 规整性：相同类型的编码段应具有相同的规定，如同类源和目标操作数尽量作同样的规定，对寄存器的使用应尽量不加约束条件。
3) 可扩充性：代码要保留一定的空间为今后的发展留一条后路，以备功能扩展。
4) 对称性：尽可能保持源操作数和目标操作数对称，这样有利于提高编译的效率。

8. 操作数的内涵

- 指令操作数所在位置的说明
- 指令直接操作的数据本身。

9. 操作数出现的形式

- 数据地址编码形式
- 立即数形式

10. GPR 型计算机的多寄存器优点与缺点

- 优点：
  1) 减少 CPU 与存储器之间的传送，加速程序的运行。
  2) CPU 内部支持过程参数传递、过程调用中的保护和传递变量。
  3) CPU 内部支持多任务前后的转换与中断处理。
  4) 支持芯片内的数据堆栈或队列。
  5) 提高芯片的规则度。
- 缺点：
  1) 寄存器越多，访问时间越长。
  2) 采用窗口指示字，寄存器地址译码时间增长。
  3) 寄存器组占用芯片面积增加。
  4) 窗口策略越灵活，CPU 的控制逻辑越复杂。
  5) 先进的编译技术相对于较小的寄存器组更有效。

6) CPU 内的寄存器在过程调用中是根据程序执行的上下关系来保护的，寄存器组越大，存储时间越长。

11. 操作数的类型

整型数(integer)、浮点数(floating point)和字符型(char)。

又具体分单字节(8 位,B,包括字符)、半字(16 位,H)、字(32 位,W)、单精度浮点、双精度浮点(2 个字,2W)。

12. 设计计算机硬件处理数的大小考虑

- 应用对数据运算精度的要求
- 数据表示形式
- 数据的实际使用率

13. 什么是寻址方式

根据操作数指示计算出逻辑地址和物理地址而得到操作数的方法，称为寻址方式。

14. 主存储器编址方法

- 第一种为低位收尾(little endian)，其字节次序是低字节在最低位的排列；（DEC、Intel）
- 第二种为高位收尾(big endian)，其字节次序是高字节在最低位的排列（IBM、摩托罗拉）

15. 对齐方式分类

- 外部对齐是指 CPU 和外部器件交换时的数据对齐关系。
- 内部对齐是 CPU 内部数据总线和寄存器之间的数据对齐关系。

16. 数据存放角度而言的地址类型

- 存储器地址：这是最主要的一组地址，指令、数据和条件控制信息大多是存放在存储器中。
- I/O 地址：这是和外部设备进行数据交换和控制处理的窗口，一般 ALU 指令不涉及到此类地址，因为外设速度总是缓慢的。
- 寄存器地址：寄存器(位于 CPU 内部，有编号或专用名称，这就是它的地址)操作速度最快，由于寄存器的数量很少，在实际应用中往往是不自觉得使用而忽略了其地址。

如果不加特殊说明，我们称地址就是指存储器地址或 I/O 地址。

17. 编址方式的分类

- 统一编址：是把除外存以外的各种存储部件的地址统一编成二进制码。如主存、通用寄存器、各种 I/O 接口寄存器等。
  - ◆ 优点：减少了指令的种类，而且访问内存的各种寻址方式和操作类型都可以应用到其它的存储单元上。
  - ◆ 缺点：一方面占用了存储器的有效地址空间；另一方面将一些控制功能转嫁到地址形式上，特别是给一些专用寄存器的编址带来麻烦。
- 单独编址：又称局部编址，是对各种不同的存储空间分别编址。
  - ◆ 优点：局部空间的编码可以得到充分利用。
  - ◆ 缺点：增加了指令种类。
- 隐含编址：多用于一些专用的存储单元。如指令计数器、程序状态字、堆栈、专用累加器等。

18. 什么是有效地址

指令中使用存储器地址说明时，寻址模式精确说明的存储器地址称为有效地址。

19. 常用寻址方式

1) 寄存器寻址
2) 立即数殉职
3) 位移和基址
4) 寄存器延时或间接寻址
5) 变址寻址
6) 直接寻址或绝对寻址
7) 存储器间接寻址或存储器延时寻址
8) 自增/自减
9) 比例寻址

20. 编译程序的功能

编译程序是将高层语言编写的源程序转换成某种机器结构的硬件能直接识别运行的并和源程序等价的目标代码(机器语言)。

21.完整编译器的组成部分

完整的编译器一般由以下几个部分组成：词法分析、语法分析、中间代码生成、代码优化和代码生成，以及相应的符号管理和出错处理。

22. 编译器的工作目标

1) 正确性——所有的应用程序必须正确的编译。
2) 编译后代码的速度。

23. 描述编译转换的过程

高层的、抽象的描述形式，逐步转换成低层的具体的表示形式，最终到达指令集。

24. 编译优化的种类

1) 高层优化——通常在源代码上进行。
2) 局部优化——只使用现行代码段优化代码(basic block)。
3) 全局优化——通过条件转移扩展局部优化，并引进变换目标集优化循环操作。
4) 寄存器优化——寄存器分配算法。
5) 机器相关优化——根据特定计算机结构的优点进行优化。

25. 高层软件存放数据的格式

1) 堆栈(stack)　用来分配局部变量。数据寻址是用堆栈指针给出地址，是一个标量而不是阵列(数组)。用于活动记录，不作为一个表达式值。
2) 全局数据区(global data area)　用于分配静态对象，诸如全局变量和常数。这种对象大部分是数组和其它数据结构类型。
3) 堆(heap)　用于分配动态对象。对象的寻址不遵守堆栈规则，堆的数据访问是用一组指针，一般不是标量。

26. 编译器优化后的结论

1) 优化没有改变指令使用率的比率。
2) 优化后总指令减少，主要是减少了 Load/Store 存储器访问指令和 ALU 指令的数目。
3) 优化后控制流指令的绝对数减少较少，变化不大。
4) 优化后控制流指令的相对百分比增加。

27. 操作系统的目的

1) 方便用户，是用户和系统结构之间的界面(编译是程序和系统结构的桥梁)；
2) 提高系统结构资源利用率，管理好资源分配和回收，合理地组织计算机系统的工作流程(包括硬件和软件)，使各种资源能有效、协调地工作。

系统结构为操作系统奠定了物理基础，而操作系统使系统结构各部件的潜力得以充分发挥。

28. 现代操作系统分类

- 批量处理系统->应用于大型科学计算。
- 单用户交互系统->用于软件开发和办公室自动化。
- 分时操作系统->将中央处理器的时间轮流分给各个联机用户的，用于多用户交互式系统。
- 实时操作系统:
  - 实时控制系统->用计算机系统对某一事件的过程进行监测控制。
  - 实时信息处理系统->接收来自终端设备的服务请求，在短时间内对用户作出正确回答。
- 网络操作系统->网络中的各台计算机配有各自的操作系统，网络操作系统把它们有机地联系起来，提供各台计算机之间的通信和实现网络资源共享或运算共享。
- 分布式操作系统->系统中的处理器在容量和功能上不尽相同，可以是微机、工作站、小型机和大型机和大型通用计算机系统，都有自己的局部存储器。分布范围大的可利用电话线通信，范围小的可利用高速总线、同轴电缆、光纤、无线信道等传输介质进行通信。

29. CISC 设计思想

1) 指令愈丰富功能愈强，编译程序愈好写，尤其认为 m-m 操作的指令效率愈高。
2) 指令系统愈丰富，愈可减轻软件危机。
3) 指令系统丰富，尤其是存储器操作指令的增多，可以改善系统结构的质量。

30. CISC 的问题

1) 指令系统庞大，一般在 200 条以上并且指令的功能异常复杂，
2) 指令操作复杂，使其执行速度下降，甚至比几条简单指令组合还慢。

3) 由于指令系统庞大，使得高级语言的编译程序选择目标指令的范围很大，从而难以优化编译，生成真正高效的机器语言程序，同时编译程序本身太长、太复杂。
4) 指令的使用率不高，且相互差别很大，其中一部分指令的利用率不到 1%。根据测量统计 90%的程序只使用了指令集中 10%的指令。

31. RISC 规律和精髓

“Simple is fast”和“Small is fast”，即：简单事件可以更快速处理；小规模器件的速度可以做的更快.

32. RISC 设计思想

指令集设计时根据 Amdahl 定律选择使用概率高的指令构成指令集，这些大概率指令一般是简单指令，因此控制器可以设计的简单、高速，且占 CPU 集成电路芯片的面积少，空出较多的集成电路芯片面积用来增加寄存器数量。在编译的配合下减少访存次数，减少指令间的各种相关和竞争，尽可能得到最佳指令序列，从而提高计算机系统的整体性能。

33. RISC 设计原则

1) 指令系统中指令选择以指令实际使用概率为主要依据，在此基础上增加了少量能有效支持操作系统、高层软件和应用领域中最有效的指令。
2) 指令结构简单寻址方式少。指令长度一般为固定，等于 CPU 字宽。编码格式固定，译码格式统一。
3) 采用 L/S 的通用寄存器(GPR)结构。
4) 尽量使 CPI=1。
5) 采用硬连线控制电路，少量采用微程序控制方式。
6) 增加通用寄存器个数，一般不少于 32 个寄存器。
7) 优化编译，提高计算机整机性能。
8) 性能评价尺度是代码的执行效率，而不是存储效率。

34. RISC 主要技术

1) 在控制逻辑上采用硬连线实现和微程序固件相结合的技术。
2) 在 CPU 中设置数量较大的寄存器组，并采用重叠寄存器窗口技术。
3) 指令技术采用流水线和延迟转移(delayed branch)技术。
4) 编译系统采用优化设计技术。
5) 存储器层次结构体系。
6) 存储器分段管理体系。

第三章 CPU 的设计

1. 处理器内部部分

- 数据通路(执行算术逻辑运算，完成计算机的计算功能)由算术逻辑运算部件 ALU(arithmetic logic unit)和一些寄存器构成，为处理器工作时数据实际流过的路径。
- 控制器(解释计算机机器指令代码，并按这些代码发出控制信号控制数据通道的工作以完成指令)是处理器中的主控部分，是将指令转换为实际硬件动作的桥梁，设计最复杂。

2. 数据通路的定义

数据通路是处理器中处理数据的部件，数据从主存取出，经数据通路的处理，得到所需要的结果后，再送回主存存放。整个数据通路的执行受控制器的控制，实际上也就是受指令的控制。

3. 数据通路的组成

数据通路由一个算术逻辑运算部件（ALU）和一些寄存器（PC、IAR、MAR、MDR、Temp 和通用寄存器 Ri 等）组成，并包括两者之间的通讯渠道（内部总线——源总线 S1、S2 和目的总线 Dest）。

4. 专用寄存器介绍

- 专用寄存器，为了完成处理器某些特定功能而设置，不用自由的用于数据计算。
- PC(Program Counter)　程序计数器：是一个专用的寄存器，存放当前所执行指令的地址，供控制器读取指令时使用，同时随时改变存放的内容以便使程序能按序执行下去。
- MAR(memory address register)　存储器地址寄存器：存放访问主存的地址。
- MDR(memory data register)　存储器数据寄存器：存放从主存取回的数据。
- IAR(interrupt address register)　中断地址寄存器：专为中断所使用。
- Temp　暂存寄存器：数据访问中起暂存作用的寄存器。
- 通用寄存器是能被用户自由地用于数据计算的寄存器(在许多计算机中以R1，R2…，Rn 来命名)。
- 寄存器堆(register file)由多个通用寄存器合起来的。存储器层次结构中的最高层，属于最小也是最快的暂存部件。

  其中，MAR、MDR、Temp 是透明的，PC、IAR 对用户是透明的。

5. ALU 与寄存器的区别

ALU 是一种非记忆的逻辑部件，其输出完全依赖于当前的输入；寄存器有内部状态，其输出由当前输入和内部状态共同决定，过去的输入会影响内部状态，从而使寄存器具有记忆。

6. 程序状态的决定

程序的状态(state)是由通用寄存器、一部分专用寄存器和标志寄存器 flag 加在一起组成的。

7. 指令执行的步骤

一条指令的执行分为以下五个步骤：

1) 取指令(instruction fetch) 。
2) 指令译码/寄存器读出(instruction decode/register fetch)。
3) 执行/有效地址计算(execution/effective address)。
   - 访存指令(access instruction)
   - ALU 指令(ALU instruction)
   - 无条件转移/条件转移指令(jump instruction / branch)
4) 存储器访问/完成转移(memory access/branch completion)。
   - 访存指令(access instruction)
   - 转移指令(branch instruction)
5) 写回(write-back）。

8. 硬连线的缺点

硬连线的控制复杂度和其状态数和输入/输出数是有关的，且随着状态数的增加，输入数也随之增加，因而使复杂度成倍增长。从而给逻辑设计、状态指派和输出表达式简化等带来很大困难。

9. 微程序控制的基本思想

微程序的基本思想是编写固化在控制器里的“微程序”，为每条指令码，编写一段程序来实现它的功能。

10. 微程序的编制方式

- 垂直微代码(vertical microcode)：编码指令，微指令短，所占控存少，格式紧凑，但要附加硬件电路，硬件开销较大。
- 水平微代码(horizontal microcode)：无编码指令。无需加解释电路，信号直接用于控制，时延短，硬件开销小，但微指令长度长，所占控存多，开销

较大。

11. 中断

中断(interrupt)是处理器的一项重要技术，在有某些重要事件发生时产生，用来使处理器转入对这些事件的处理。

Intel 和 IBM 仍将所有的都称为中断，而 Motorola 将它们称为例外，DEC 则根据不同的情况，将它们称作异常、出错、自陷、放弃或中断。

12. 中断的类别

中断可以按以下几种标准，分成不同的类别：

- 同步中断与异步中断，如果一个中断总是在同样数据和同样内存映象下程序运行到同一地址时发生，则称这个中断为同步中断。与它相对的是一些硬件产生的中断，如 I/O 中断，只与外设本身的状态有关，称它们为异步中断。
- 用户请求的中断和强迫中断，如果中断是由用户请求发生的，称为用户请求的中断，否则就是强迫中断。
- 用户可屏蔽中断和不可屏蔽中断，如果用户可以让处理器不理会一个中断，则称这个中断是用户可屏蔽中断。通常一些牵涉硬件错误的中断，如电源失效是不可屏蔽的。
- 在指令运行中产生的中断和指令间产生的中断，这是以中断产生的时间分类。这种中断将给控制器设计带来最大的影响。
- 可恢复执行的中断和应终止系统运行的中断。如果该中断发生就会导致程序停止运行，则称中断为终止系统运行的中断。

第四章 流水线技术

1. 流水线的作用

提高硬件功能部件的使用率，减少指令的平均执行时间。

2. 什么是流水线

流水线(pipeline)是指在程序执行时多条指令重叠进行的操作的一种准并行处理实现技术。

3. 流水线深度

流水线深度是指流水线中总的流水级的数目。

理想情况下流水线计算机的加速比(speedup)就等于流水级的深度。

4. 流水线分类

- 处理的级别：操作部件级、指令集流水、处理器级流水。
- 完成的功能：单功能流水线、多功能流水线
- 连接方式：静态流水线、动态流水线
- 处理的数据：标量流水线、向量流水线
- 流水线结构：线性流水线、非线性流水线

5. 流水线竞争的种类

- 结构竞争(structure hazard)(资源竞争)：由资源缺乏引起。
- 数据竞争(data hazard)：由指令间数据相关而引起。
- 控制竞争(control hazard)：由程序指针 PC 值的改变而引起。

6. 什么是结构竞争

如果硬件资源不足而使某些指令组合不能实现，这就产生了结构竞争。

7. 结构竞争产生的原因

1) 某些功能单元没有完全流水
2) 某些硬件资源数量不足，造成同一个时钟周期某些指令组合争抢硬件资源

8. 结构竞争的处理方法

当流水线碰到存储器数据存取时，流水线暂停一个时钟周期，即插入一个停顿（Stall）。

9. 为什么要允许结构竞争

- 减少成本。
- 降低单元电路的延时时间。
- 减少电路的复杂程度。

10. 什么是数据竞争

数据竞争（data hazard）由指令间数据相关而引起。某条指令的执行依赖于前面指令的执行结果

11. 防止产生数据竞争的方法

可采用旁路的硬件技术来消除由前面这段指令所产生的数据竞争。旁路技术也称提前(forwarding)电路或短路电路(short-circuiting)

12. 数据竞争的类型

- 先写后读相关 RAW (read after write) （写读冲突）
- 写写相关 WAW (write after write) （写写冲突）
- 先读后写相关 WAR (write after read) （读写冲突）

13. 指令发射的概念

指令从 ID 级流入 EX 级，一般称指令发射(instruction issue)。一条指令已建立了这一过程，称为已发射(issued)。

14. 产生控制竞争的原因

控制竞争的原因主要来自转移指令。

15. 几种常见的流水转移方案

1) 预测转移成功
2) 预测转移不成功
3) 提前生成条件码
4) 适当增加指令缓存器
5) 提高转移预测命中率
6) 延时处理技术
7) 硬件支持延时转移

16. 流水线计算机处理中断

1) 强制调用自陷指令，即下一取指级 IF 级取 Trap 指令进入流水线。
2) Trap 指令中止所有指令的写操作，包括引起中断的指令及所有后继的进入流水线的指令。
3) 操作系统的中断处理例程接管控制后，该例程立即保存故障指令的 PC 指针，用于中断恢复后的返回。如果我们用了延时转移技术，那么只保存一个 PC 指针是不能完全恢复机器状态的，因为此时流水线中的指令是不连续的。因此考虑延时转移中断时，要保存一组 PC 指针，一般是保存（转移延时槽数+1）个 PC 指针。中断结束后，重新取出保存的 PC 指针组，流水线工作从中断子程序返回，恢复原来的指令流。

17. 精确中断的概念

如果中断时，流水线正好停留在引起异常中断指令前的指令已经执行完成之时，而后面的指令能从断点处重新启动，因此我们称此流水线为精确中断(precise interrupt)。

18. 现代高性能计算机的中断模式

一种是采用精确中断，另一种是采用不精确中断。

19. 流水线中断处理的原则

- 如果多个异常中断顺序出现，常用的方法是先处理与最早指令有关的异常情况，以避免指令相关性；
- 如果多个异常同时出现，那么应首先处理最严重的异常中断。如某指令 i 在 EX 级发生溢出，而指令 i+1 在 ID 级发现非法指令代码，那么应首先处理溢出；
- 流水线中某条指令前几级流水级出现异常中断，该指令前的指令必须先执行完，避免前指令对其产生影响，然后再进行异常中断处理。

20. 指令交付的概念

当一条指令确保一定能完成我们称其为交付(committed)。

21. 功能部件的等待时间

功能部件的等待时间(latency)是产生结果指令和使用该结果的指令之间必须间隔的时钟数；

22. 功能部件的启动间隔或重复间隔

功能部件的启动间隔或重复间隔(initiation interval 或 repeat interval)是指两个进入同一部件的操作之间所必须等待的时间间隔。

23. 扩展流水线机器的预防及检测竞争出现

- 浮点除法部件没有用流水方式实现，故有结构竞争产生，需要对其进行检测；若有竞争，则加入 Stall 延时发射浮点除法指令。
- 由于不同指令的执行时间是不同的，因而在一个时钟周期内要回写寄存器的操作会多于一个。

- 可能出现 WAW 型的数据竞争。因为有可能后面的浮点指令执行的时间短，可以先到达 WB 级，而前面的浮点指令执行的时间长，后到达 WB 级。由于读寄存器操作总是在 ID 级，故不可能出现 WAR 型数据竞争。
- 指令执行完成的顺序和它们被发射时的顺序是不一致的，我们称为“乱序”。
- 由于有较长的操作等待时间，用 Stall 消除 RAW 竞争的概率增加。

24. 流水线内所以及检测资源竞争的两种方法

1) 在 ID 级跟踪写端口的使用情况，并在指令发射前插入 Stall。
   - 优点：保持了仅在 ID 级使指令暂停移位的特性。
   - 缺点：要增加一个流水寄存器以及相应写冲突控制逻辑。
2) 在冲突指令试图流入 MEM 级时插入 Stall，暂停冲突指令。
   - 优点：不需要在 MEM 级以前检测冲突，而在 MEM 级或 WB 级可以方便地看出回写寄存器端口的竞争。
   - 缺点：流水线控制复杂，流水线停顿将出现在两个地方。因为在 MEM 级之前插入停顿周期会使 EX 级、A4 级、M7 级被占用的时间也向后推迟，因此可能再次造成竞争，又必须再次插入 Stall。

25. 处理 WAW 竞争的方法

1) 延时发射 LD 指令，等到 ADDD 指令进入 MEM 级，才发射 LD 指令。
2) 检测出 WAW 竞争，并消除 ADDD 指令的结果，即改变控制，使 ADDD 指令不写操作结果。

26. 指令发射前应该进行的检测

1) 检测结构竞争。检测所需功能部件是否空闲，当有指令需要写端口，而且此写端口是空闲的（可用的），则发射当前指令。此处只有非流水除法部件才需要。
2) 检测 RAW 数据竞争。 检测当前指令的源寄存器操作数是不是已在流水线

中指令的目的操作数。如果不是，或目的操作数正确结果值已在流水寄存器中，并可以作为源操作数直接旁路引用，则发射当前指令。

3) 检测 WAW 竞争。检测任何在 A1，A2，A3，A4，DIV，M1，M2，…，M7 的指令的目的操作数和当前指令的目的操作数是否相同，如果相同，则暂停发射当前指令。

27. 乱序完成的概念

在 ID 级先发射的指令，有可能要在后发射的指令完成之后才完成。

28. 处理乱序执行的常用方法

1) 忽略乱序执行造成的影响。
2) 用缓冲器保存指令操作的结果，一直到该指令前发射的操作都完成为止。
3) 允许中断具有一定的不精确性，仅保存足够的信息，使自陷处理子程序能建立异常中断的精确序列。
4) 采用流水线指令发射暂停方法。

29. 流水线动态调度的概念

通过硬件在程序执行时重新安排代码的执行序列来减少竞争引起的流水线停顿时间，我们称之为流水线动态调度。

30. 动态调度的主要思想

在程序执行的过程中适时进行恰当地调度以消除流水线竞争。

31. 动态调度的优点和缺点

优点：

- 能调度在编译时不可能知道的竞争情况。
- 符合程序执行的实际情况。

- 具有更高的效率和准确性。
- 简化编译程序的设计。
- 代码的移植性强。

缺点：控制硬件较复杂。

32. 记分牌动态调度算法（集中式动态调度）

记分牌(score boarding)就是一种允许指令乱序执行，克服资源和数据冲突的技术。

33. 记分牌算法的目的

记分牌的目的是在无资源竞争的前提下保持每一个时钟周期执行一条指令的速率，尽可能提早指令执行。当一条指令暂停执行时，如果其它后继指令与暂停指令及已发射的指令无任何相关，则仍然可以发射、执行。

34. 记分牌算法的任务

记分牌的任务是控制指令的发射和执行，同时也包括检测所有竞争。

35. 记分牌的流水步骤

1) 发射级(issue) //*处理结构竞争和 WAW 竞争
2) 读操作数(read operands) //*动态解决 RAW 竞争
3) 执行(execution)
4) 写结果(write result) //*处理 WAR 竞争

36. 构成记分牌的三个部分

1) 指令状态。指出指令工作处在上述四级中的哪一级。
2) 功能部件工作状态。指出功能部件的工作情况，每个功能部件需要指出九项相关参数。

- Busy——指出功能部件地忙或空闲状态。
- Op——功能部件所执行的操作类型。
- Fi——目的寄存器。
- Fj，Fk——源操作数所用的寄存器。
- Qj，Qk——产生源寄存器数据的功能单元。
- Rj，Rk——指示源寄存器 Fj，Fk 准备就绪。

3) 寄存器结果状态。如果有一条已激活指令有一个目的操作数是寄存器，则指出那个功能单元将写（操作）这个寄存器。

37. 记分牌消除 Stall 收到的因素限制

- 指令间可用于并行操作的数量：这个因素决定了可以找出多少不相关的指令来执行。
- 记分牌记录项目数：此因素决定了流水线能向前处理不相关指令数的程度。可进入记分牌等候执行的指令集合称为窗口。
- 功能部件的类型和数量：此因素对避免结构竞争很重要。在动态调度中要求增加这些功能部件。
- 逆相关(antidependence)和输出相关(output dependence)：如果读指令 i 和写指令 j 之间，指令 i 先激发而指令 j 后激发，且指令 j 写的对象是指令 i 要读的源，则称两指令间存在逆相关。逆相关和输出相关导致了 WAR 竞争和 WAW 竞争，是由于指令使用相同的源和目标操作数载体引起的，也称之为名称相关。

38. 什么是转移预测缓冲器

转移预测缓冲器是用转移指令地址的低位来寻址的存储器。是最简单的动态转移预测方案

39. 转移预测缓冲器（转移历史表）的优点和缺点

- 优点：这个缓冲器是一个每次都命中的高效 Cache，而采用缓冲器后的性能

改进取决于转移预测和程序中转移的行为关系如何以及正确的预测是如何进行比较的。

- 缺点：虽然转移几乎总是发生的，但当出现不发生的转移指令时，会出现两次不正确预测。

40. 什么是两位预测方案

两位预测方案，即出现两次预测失误才改变预测器状态。

41. 如何提升预测正确率

1) 增加预测缓冲器容量；
2) 改进预测方案，提高预测度。

42. 什么是指令集并行性

要进一步降低 CPI，提高计算机的性能，就必须挖掘指令间存在的不相关性和并行处理的潜力，我们称之为指令级并行性（instruction-level parallelism，ILP）。

43. 什么是循环级并行

转移指令中有很大部分是循环转移引起的，最简单和最普遍的方法是开发循环迭代之间的并行性，这种典型的并行性处理称为循环级并行(loop-level parallelism)。

44. 什么是循环展开

将循环体多次复制，增加与转移相关的指令进入循环体并调整循环体终点代码，可以减少前述三个时钟周期的开销，这种方法称为循环展开(Loop unrolling)。

45. 什么是循环传递相关

假定某一时刻只有一种相关存在，那么第一种相关中本次 S 语句需要前次迭代的 S 语句结果，它是循环传递相关。

46. 什么是可并联的

当一个相关性不是周期环绕的，我们称它是可并联的。

47. 多发射处理器的种类

多发射处理器有两种类型：一种是超标量处理器(superscalar processor，SP) ，另一种是超长指令字(very long instruction word，VLIW)处理器。前者是静态+动态，后者是静态结构。

48. 什么是超标量处理器

在一个时钟周期发射多条指令进行并行处理的流水线技术称之为超标量流水线处理器。

49. 超标量技术的优点

- 受代码密度影响小，处理器只要检测下一条指令是否能发射，不必调度更多指令去配合发射能力；
- 不进行编译优化调度程序或用旧编译实现的目标程序也能运行，只不过运行性能差些。要克服这些缺点可以采用动态调度。

50. 什么是超长指令字处理器

编译器根据程序指令级的并行性，进行优化调度，按一定的处理要求生成多个可以同时执行的操作，并将其作为一个整体指令格式或指令包。我们把能一次同时发射执行上述多个操作的策略称为长指令字或超长指令字(very long instruction word，VLIW)处理技术。

51. 多指令发射器的限制

提高指令发射率主要受到三方面的限制：

1) 程序中固有的指令级并行性的限制。程序中可利用的 ILP 是最直接最基本的限制。由于程序中转移指令的概率很高，在以静态调度为主的处理器中如果不采用循环多次展开，就没有足够的并行操作来填入处理器的发射槽。
2) 多发射处理器实现基本硬件的复杂性。同时发射和执行多条指令使多发射处理器的硬件资源复杂，代价过高。
3) 超标量或 VLIW 特有的限制。采用超标量和 VLEW 技术实现多发射还有其固有问题。

52. VLIW 技术的一些问题：

1) 代码膨胀。VLIW 技术所用的两种技术使代码剧增：为了提高并行性，需要产生足够的线性代码编译展开循环，从而增加了代码长度；VLEW 的固定指令格式为了使进入处理器的指令不冲突，在封装指令时不能填满不使用的功能单元，这形成了浪费位，也增加了代码长度。
2) 锁定同步操作。VLIW 采用静态调度封装指令，然后同时发射，这就需要锁定同步操作。
3) 二进制代码兼容性。这是 VLIW 处理器的主要逻辑障碍。这个问题发生在同一代系列处理器中。
4) 多发射处理器和向量机关系。多发射处理器的特点是力求利用大量的指令级并行性。多发射处理器潜在的优点有：第一，多发射处理器可以利用某些规律性不强的代码结构中的并行性；第二，VLIW 存储系统较向量机便宜。

53. 如何开发程序中指令级的并行性

1) 寻找线性基本块中的不相关代码，用于编译调度。

2) 寻找那些可能包含有并行性的循环。

3) 消除变量名相关（name dependence）。

54. 什么是软件流水

软件流水是从源循环的不同迭代体中取出必要的指令，重新建立新的循环，提供连续指令给多发射处理器。

55. 软件流水的具体流程

根据循环体的特点展开若干次，然后从展开后的各迭代中分别取出部分指令作为新的循环体，吸取指令的条件是新循环体内无数据相关（有最大的指令级并行性）。然后将剩下的指令分成两部分，前一部分构成启动代码，最后一部分构成结束代码。

56. 软件流水的优点

节省代码空间，它减少每次循环迭代都出现的影响循环体最大运行速度的因素（启动代码和结束代码），把它们排出循环体外只运行一次。

57. 程序中转移指令出现的场合

循环调用、无条件控制程序指令流向和根据要求控制程序指令流向。

58. 什么是路径调度

路径调度的基本思想是基于转移预测，在程序中有很多转移指令的转移行为，在一定程度上是可以预测的，于是我们在编译时，首先预测代码执行的方向，即执行路径。如果这条路径使用的概率是较大的，那么我们将这条路径上的两个基本块合并成一个基本块，从而扩大了基本块。这突破了转移指令对线性代码基本块的限制，增加了更多的指令级并行性潜力。

59. 路径调度的两个独立过程

1) 路径选择：首先根据转移行为预测转移可能的两个路径方向，找出使用概率大的那个方向作为扩展基本块的方向，这个方向的后继指令称为预测路径(trace)。
2) 路径压缩：其目的是将选定路径上的操作封装成数量较少的长指令或超长指令。路径压缩尽可能将路径上的操作移动到转移指令的前面或后面，但又不影响转移的封装相关性，这个过程实际上是代码序列重新调度过程。路径压缩是一种全局代码调度，数据相关和控制相关妨碍了这种压缩调度。我们采用转移行为预测和调度补偿来消除控制相关影响。

60. 路径调度的补偿结构

- 转移出去，称作分离(split)，实际上这个点是调度路径的入口
- 转移进入被调度的基本块，称作重接(rejoin)，这一点是一个基本块的出口。

我们用两个模块来重新连接调用路径：即分离模块和重接模块。这两个模块的作用是在预测失效时消除路径调度和压缩的影响。

61. 路径调度的优缺点

路径调度的主要优点是可以跨越转移点进行多基本块的调度，在预测成功时，可以开发出更多的并行性供多激发处理器尤其是 VLIW 处理器运行，有效地提高了机器性能。

但当预测失败时，由于要消除跨基本块调度的影响，需要执行更多的指令来重新建立和外路径的连接，反而使运行性能下降，而且路径调度越成功，失败补偿就越困难，需要的代价也越大。

62. 在硬件支持下克服编译技术扩展指令级并行性的方法

- 扩充指令系统，即增加条件指令(conditional instruction)或预测指令(predicated instruction)。这种指令可以消除转移并帮助编译跨越转移指令进

行调度指令。但利用条件指令或预测指令开发的指令级的并行性仍然有限。

- 这种技术的主要思想就是投机，即在处理器知道哪条指令应该执行以前，先投机地执行一条指令（即避免控制相关的等待），下面有两种投机方法：
  1) 在硬件支持下由编译完成的静态投机，这种方法是编译选择一条指令使它投机执行，而硬件在投机失败时帮助忽略投机指令的执行结果，条件指令也可以用来实现有限的静态投机。
  2) 硬件动态投机，这种方法是通过转移预测来指导投机过程。

63. 什么是条件指令

指令操作附带一个条件，指令的执行包括条件的计算并根据条件真假来决定指令是否执行，如果条件为真，则指令正常执行；否则，指令看作是一个空操作。

最常用的条件指令是条件传输指令，即当条件为真时将数值从一个寄存器传送到另一个寄存器。

64. 条件指令的作用

条件指令的作用是把控制相关转换成基于转移条件条件码(R1)的数据相关

65. 正确使用条件指令的前提

要正确使用条件指令，我们必须保证投机指令本身不会引起异常中断。

66. 条件指令使用的限制因素

1) 当条件为假时，取消条件指令还是要花费时间。
2) 条件指令的使用还受到条件码的影响。
3) 当控制流包含两个以上的转向分支时，条件指令的使用会受到限制。
4) 条件指令与无条件指令相比可能会有速度损失，因为条件指令的执行可能需要较多的时钟周期数或较长的时钟周期。

67. 支撑编译采用更大胆地投机调度的方法

- 硬件和操作系统协同忽略投机指令引起的异常中断。
- 设置一组状态位（抑制位），当投机指令引起异常中断时，投机指令的结果寄存器就附加一个抑制位。当正常指令企图访问这个寄存器时，抑制位就产生一个错误信息。
- 用硬件指出某条指令是投机的，并且用硬件缓冲存储结果，一直到处理器确定指令的执行不再是投机行为为止。

上述这些方案使用时，必须区分两种异常中断：

- 程序错误，一般会导致程序终止运行；
- 异常中断处理后，可以恢复正常运行。

68. 三种常用的硬件支持的编译投机调度方案

- 硬件软件协同投机。
- 采用抑制位的投机技术。
- 基于改名的投机指令。

69. 什么是超级流水线(superpipeline)技术

一个处理器具有较高的时钟频率和较深的流水级(如 8 级、10 级)，称之为超级流水线(superpipeline)技术。

70. 实现超级流水线的方法

增加流水深度的主要方法是将原流水线的功能部件流水化。超级流水线技术的实现方式一般是将基本流水线，如第一代 RISC 的流水线中的若干流水级进一步分成两个小过程或更多的过程，并在一个机器时钟内发射多条指令，在一定的流水线调度和控制下，使得每个小过程和其它指令的不同小过程并行执行，从而在形式上好像每个流水周期都可以发送一条指令。因此超级流水线结构的处理器，机器时钟和流水线时钟是不同的。

第五章 存储器的层级结构

1. 评价存储器性能的参数

1) 容量：S=W×l×m 表示。其中 W 为存储器字长，l 为存储器字节，m 为存储器个数。
2) 速度：访问时间(access time)Ta：从存储器接到读请求到所读的字传送到数据总线上的时间间隔。
   - 存储周期 Tm：连续两次访问存储器之间所必需的最小时间间隔。一般，Tm> Ta 。
   - 存储带宽 Bm：存储器被连续访问时所提供的数据传输速率，单位是位(或字节)／s。
3) 价格：通常用单位字节价格来表示。若总容量为 S 的存储器的总价格为 C，则单位字节价格 c = C／S。

2. 存储体系的设计目标

高速度、大容量、低价格

3. 什么是块

块(block)：相邻两级存储器之间信息交换的最小单位。块大小一般是固定的，但也可以是可变的。若块的大小固定，则两级存储器的容量为块大小的整数百倍。

4. 什么是命中率

CPU 产生的有效地址可以在高层存储器中访问到的概率。

5. 什么是失配率

失配（效）率(miss rate)M：CPU 产生的有效地址不能直接在高层存储器中

访问到的概率。M=1－H。

6. 什么是命中时间

命中时间(hit time)HT：访问到高层存储器所需的时间，其中包括本次访问是命中还是失配的判定时间。

7. 什么是失配损失

用低层存储器中的相应的块替换高层存储器中的块，并将所访问数据传送到请求访问的设备（通常是 CPU）的时间。又可细分为访问时间和传送时间。

8. 什么是访问时间

访问高层存储器失配时，在低层存储器中访问到块中第一个字的时间（又称访问延迟 access latency)，它与低层存储器的延迟有关，而与块大小无关。

9. 什么是传送时间

传送块内字的时间，它依赖于两级存储器之间的传输带宽和块大小，一定传输带宽下与块大小成正比。

10. 什么是平均存储器访问时间

平均存储访问时间(average memory_access time,AMT) 更好地评价存储器层次结构的性能参数是平均存储的访问时间，平均存储访问时间＝命中时间＋失配率×失配时间表示为：AMT＝HT＋M×MP

11. 块大小和失配率之间的关系

- 当块大小过小时，失配率很高。随着块大小的增加，由于有效地利用了程序的空间局部性，失配率呈现下降趋势。
- 当高层存储器容量保持不变时，失配率有一最低限值，此时块大小的变化

对失配率没有影响。

- 当块大小超过某定值（又称瑕点）后，失配率呈现随块大小增加而上升的趋势。

12. 设计存储器层次结构的根本目标

设计存储器层次结构的根本目标是为了减少执行时间，因此在确定块大小时，不能以失配率为标准，而应选择使平均存储访问时间最小的块大小。

13. 计算机设计的最终目标

处理器的性能是计算机设计的最终目标。

14. 存储器层次结构设计中四大基本问题

- 映象方式
- 映象机构
- 替换(算法)策略
- 写(算法)策略

15. Cache 映像方式

- 直接映象(direct mapped)：这是最简单的一种映象方式。主存中的一信息块只能对应 Cache 的一个特定行。
- 全关联映象(fully associative)：主存中的一信息块可对应 Cache 中的任意一行。
- 组关联映象(set associative)：将 Cache 的行分成若干组，不妨设为 G 组，G＝2S，则每组中有 n＝2Cb/G＝2(Cb-S)＝2e 行。主存中的第 i 块可以对应 Cache 中的某一特定组中的任意一行。若组中有 n 行，则称之为 n 路组关联映象。

16. Cache 命中判定过程

当 CPU 给出一地址后，首先由索引确定 2s 组中的一个组，然后地址标志和该组中的所有行标志（共 2e 个，每个 Mb-S 位）进行相联比较。若有相同的且有效位被置位，则表示相应信息块在 Cache 中，命中 Cache 后由块内偏移地址确定要访问的具体字节。否则，Cache 失配。

17. Cache 替换策略

- 随机替换策略(RAND)
- 先进先出策略(FIFO)
- 最近最少使用策略(LRU)

18. Cache 写策略

- 直写（write through）：信息被写入 Cache 行的同时，利用 CPU 和主存之间的直接数据通路写入主存的对应块中。
- 回写（write back）：信息只写入 Cache 的相应行，仅当被修改过的行被替换出 Cache 时，才将它写入主存的相应存储块中。

19. 回写技术的优点

- Cache 命中时，写操作是以写 Cache 的速度进行，而且在一个数据块内的多次写操作只需要对低层存储器写一次。

20. 直写技术的优点

- 读操作引起的 Cache 失配不会导致对低层存储器的写操作。
- 主存中总是保存着数据的最新拷贝

21. 写失配时的修改数据载入 Cache 方案

- 写分配（write allocate）：将要写的数据装入 Cache，然后进行和写命中时相同的操作，这种方法与读失配时的处理方法类似。

- 无写分配（no write allocate）：也称为绕写（write around），直接对低层存储器中的数据块做改写操作，不再将此数据块装入 Cache。

22. Cache 读命中时的处理流程

1) 来自 CPU 的地址被分为 29 位块帧地址和 3 位块内偏移地址，块帧地址又分成 20 位标志和 9 位索引。
2) 根据索引选择 Cache 中的一个组，读取组内各行标志以判定要访问的数据块是否在 Cache 中。
3) 块帧地址的标志域与步骤 2 中读取的两个行标志作相等比较。
4) 假设有一行标志与块帧地址的标志相匹配，则由 2 选 1 多路转换器选取相应的数据行。
5) 读出的字送往 CPU。

23. Cache 写命中时的处理流程

（前四步与读命中相同）

1) 来自 CPU 的地址被分为 29 位块帧地址和 3 位块内偏移地址，块帧地址又分成 20 位标志和 9 位索引。
2) 根据索引选择 Cache 中的一个组，读取组内各行标志以判定要访问的数据块是否在 Cache 中。
3) 块帧地址的标志域与步骤 2 中读取的两个行标志作相等比较。
4) 假设有一行标志与块帧地址的标志相匹配，则由 2 选 1 多路转换器选取相应的数据行。
5) 修改数据块，并将数据更新部分写入 Cache 相应行。

24. 什么是一致 Cache 或混合 Cache

若 Cache 中既可以存放指令，又可以存放数据，此时称之为一致 Cache(unified Cache)或混合 Cache(mixed Cache)。

25. 数据 Cache 和指令 Cache 的失配率关系

指令 Cache 的失配率要低于数据 Cache。

26. Cache/主存存储器层级结构对计算机性能的影响

- CPI 越小，Cache/主存对 CPU 性能的影响越大。
- 因为主存一般是用相同的存储器芯片实现的，并且独立于 CPU，所以不同计算机的主存通常有着相近的访存时间。但在计算 CPI 时，Cache 的失配损失是以 CPU 时钟周期为单位来计算的，因而尽管主存有着相同的访存速度，较高的 CPU 时钟频率将导致较大的失配损失。
- 对于 CPU 时钟频率较高，CPI 较小的处理器，如 RISC 处理器，Cache 失配对性能的影响是极严重的。所以在设计此类处理器时必须尽量降低 Cache 的失配率，同时在评估此类处理器的性能时绝对不可忽略 Cache 失配所造成的影响。

因为降低存储器层次结构的平均访问时间是我们的设计目标，本章的大部分将用平均存储访问时间来评价存储器层次结构的性能。但是必须明确，最终的设计目标是减少 CPU 时间。

27. Cache 失配原因

- 被迫(compulsory)失配：第一次访问存储块时，由于该块不在 Cache 中，所以必须首先将此存储块从主存取入 Cache 中。这样引起的失配也称为冷启动失配(cold start miss)或首次访问失配(first reference miss)。
- 容量(capacity)失配：如果 Cache 不能容纳程序执行过程中所需的所有存储块， 那么当程序再次访问到曾装入 Cache 又已被替换出去的某存储块时，就会出现容量失配。
- 冲突(conflict)失配：在采用组关联和直接映象方式的 Cache 中，主存的很多块都将映象到 Cache 的某一行。如果因为这个原因，当程序再次访问到曾装入 Cache 又被替换出去的某存储块时，就会出现冲突失配，也称为碰撞失配(collision miss)。

28. 减少失配损失的方法

1) 加快写失配操作
2) 加快读失配操作
3) 非阻塞 Cache
4) 两级 Cache

29. 降低失配率的办法

1) 增大块大小
2) 提高关联度
3) 增设替换块 Cache
4) 伪关联映像
5) 硬件预取
6) 编译插入预取指令
7) 编译优化

30. 伪关联 Cache 的命中方法

- 快的命中: 对应的命中时间即常规的命中时间。
- 慢的命中: 对应的伪命中时间（一般伪命中时间>常规命中时间）。

31. 减少伪命中次数的方法

当发生伪命中时，由硬件交换原对应块和伪对应块，使得后续的对该块的访问均为快速命中。

32. 伪关联 Cache 的特点

既能降低组关联 Cache 的失配率，又能降低直接映象 Cache 的命中时间。

33. 硬件预取的目的

有效地降低 Cache 的失配率，而且不影响 CPU 的时钟周期。

34. 硬件预取的方式

在 CPU 申请访问指令和数据之前就把可能用到的指令和数据从存储器中取出来，预取的指令和数据可以取进 Cache，也可送入专设的预取缓冲器中。

35. 编译控制预取的方式

- 寄存器预取(Register Prefetch)——预取的数据送入寄存器；
- 缓存预取(Cache Prefetch)——预取的数据只送入 Cache.

36. 什么是非约束预取

预取不改变寄存器或存储单元的值也不会导致异常中断，这样的预取也称之为非约束(nonbinding)预取。

37. 编译优化的方向

- 降低指令访问的失配率
- 减少数据访问的失配。

38. 编译优化的常用方式

- 合并数组
- 内外循环转换
- 循环融合
- 模块化访问

39. 如何加快写命中操作

1) 对于直写 Cache，将 Cache 的写操作设计成流水工作方式。

2) 采用“子块映象方式”，使写命中操作在一个时钟周期内完成，但它只适用于直写 Cache。

40. 采用子块映象的优点

- 减小大存储块引起的失配损失；因为在失配时需要读入的只是大存储块的一部分，即子块。
- 减少小容量 Cache 的标志存储器在 Cache 总容量中所占的比例。一行包含多个子块，不必为每个子块都做标志，只需为每个子块配备一个有效位。

41. 如何缩短写命中时间

不管地址标志是否匹配，都将数据写入 Cache，置有效位，然后将更新的字写入主存的相应块。

42. 虚拟存储器的分类

- 块大小固定不变的页式虚拟存储器
- 块大小可变的段式虚拟存储器。
- 段页式存储器

43. 页式存储器和段式存储器的优缺点

- 页式存储器虚地址长 1 字节，对程序员不可见，替换容易，有内部碎片，磁盘传输效率高。
- 段式存储器虚地址长 2 字节，对程序员可能可见，较难替换，且有外部碎片，难以确定磁盘传输效率。

44. 主存和辅存间的对应关系

- 直接映象
- 组关联映象

- 全关联映象

45. 两级 Cache

CPU 的速度越来越快，主存的容量越来越大，而主存和 CPU 之间的速度差距也越来越大。因而体系结构的设计就面临这样相互矛盾的要求：一方面要将 Cache 设计得更快（意味着容量小）以匹配 CPU 的速度增长。另一方面则要将 Cache 设计得更大（意味着速度慢）以缩小主存和 Cache 之间日益增大的容量差距。两级 Cache 就可以同时满足速度和容量两方面的设计要求。即在原来的 Cache 和主存之间再增加一个 Cache。

- 一级 Cache，一般做在 CPU 芯片上(也称片内 Cache)。它容量较小，速度与 CPU 的时钟周期相匹配。
- 二级 Cache，一般做在 CPU 芯片外(也称为片外 Cache)，它容量大，速度在 CPU 和主存之间，以便尽可能多的访问二级 Cache 中完成而不必再去访问主存。

46. 主存的组织方式

- 单体单字主存结构
- 单体多字主存结构
- 多体交叉主存结构

额外

1. 仓库级计算机

Worehouse Scale Computer，仓库级计算机，是商用因特网基础的超大规模集群，许多人每日所用 Internet 服务的基础。