第二章　指令系统原理与实例

- 2.1 简介
- 2.2 指令集系统结构的分类
- 2.3 存储器寻址
- 2.4 操作数的类型
- 2.5 指令系统的操作
- 2.6 控制流指令
- 2.7 指令系统的编码
- 2.8 编译器的角色
- 2.9 MIPS系统结构
- 2.10 谬误和易犯的错误
- 2.11 结论

2.10 谬误和易犯的错误

- 易犯的错误：设计各种专门支持高级语言结构的“高级”指令。

为了支持高级语言的特征，会设计一些功能很强的指令。但是，这些指令经常会做些需求以外的工作，或者不能准确地符合一些语言的要求。

对于经常发生的情况来说，这种指令的功能通常过于强大，这就导致了许多不必要工作以及指令速度降低，也使硬件更复杂。

2.10谬误和易犯的错误

- 谬误：可以根据一种典型的程序来设计完美的指令系统。

  很多人倾向于相信存在一个典型的程序，可以用它来设计一个理想的指令系统。我们可以参考综合基准测试程序，其中的数据清楚地表明应用程序在使用指令系统方面存在显著的差别。

- 例如，如下图所示的在四个SPEC2000的程序中数据传输大小的情况：很难说这四个程序中哪一个是典型的。对于专门支持一类应用的指令系统，这种差异可能会更大，例如十进制指令在其他应用中就不会被使用。

2.10谬误和易犯的错误

- 谬误：可以根据一种典型的程序来设计完美的指令系统。

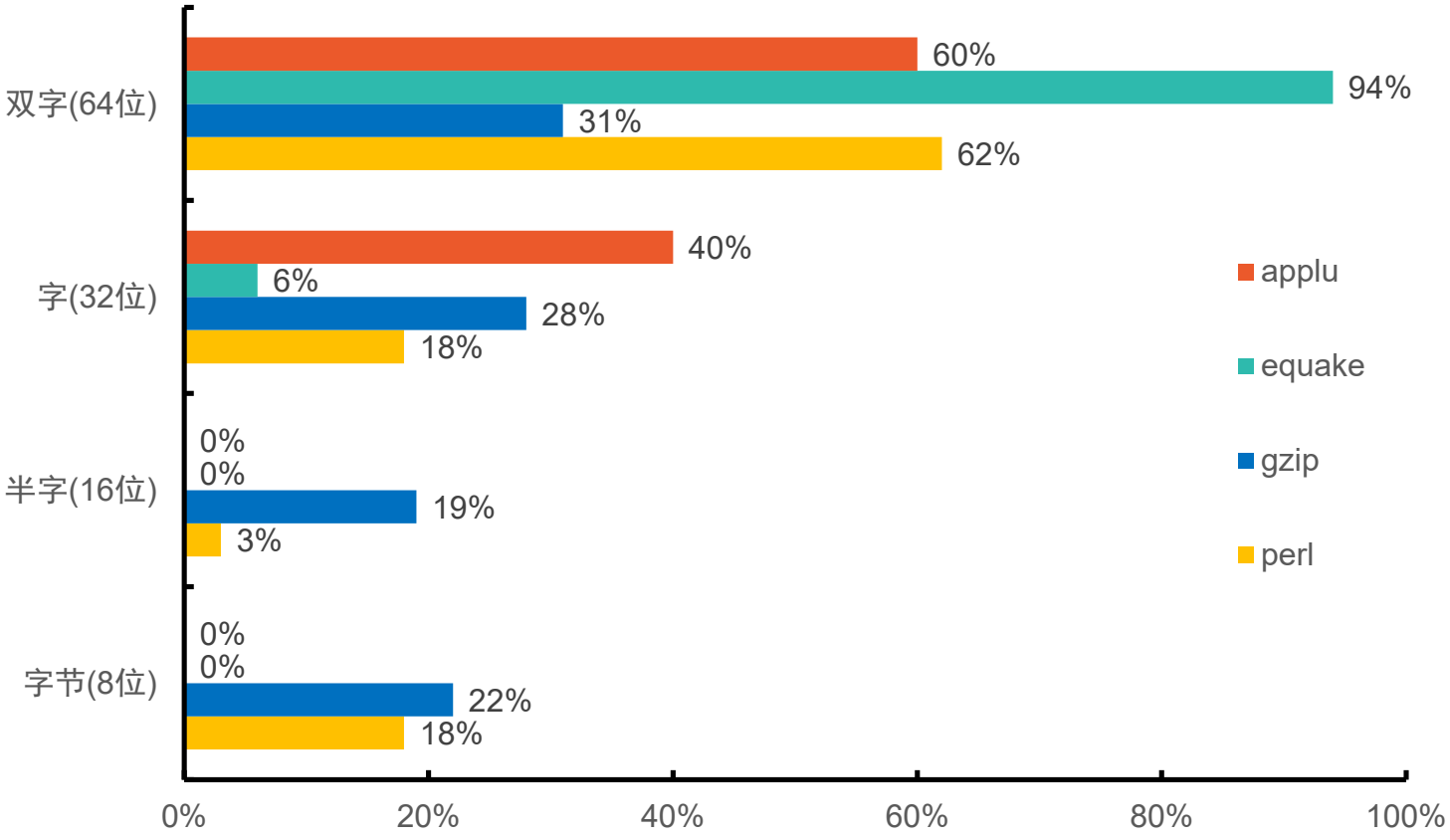

Figure A.29

2.10谬误和易犯的错误 Figure A.30

- 易犯的错误：可以不考虑编译器而改进指令系统以缩减代码大小。下表显示的MIPS指令系统下四个编译器产生的EEMBC的通信程序相对代码大小。尽管设计师努力使代码减少了30%~40%，不同的编译器策略却更大程度地影响着代码的大小。就像性能优化技术一样，在试图改进硬件以节约空间之前首先应该考虑编译器如何产生较少的代码。

编译器	Apogee software Version 4.1	Green hills Multi2000 V 2.0	Algorithmics Sde 4.0b	Idt/c 7.2.1
系统结构	MIPS IV	MIPS IV	MIPS 32	MIPS 32
处理器	NEC VR5432	NEC VR5000	IDT 32334	IDT 79RC32364
自相关内核	1.0	2.1	1.1	2.7
卷积编码器内核	1.0	1.9	1.2	2.4
定点位分配内核	1.0	2.0	1.2	2.3
定点复数FFT内核	1.0	1.1	2.7	1.8
Viterbi GSM译码器内核	1.0	1.7	0.8	1.1
5个内核的几何平均值	1.0	1.7	1.7	2.0

2.10谬误和易犯的错误

- **谬误：有缺陷的系统结构不可能是一种成功的系统结构。**

80x86典型例子：它的结构只有它的设计者才喜欢。Intel的工程师们试图更正80x86设计中不受欢迎的设计。例如，80x86支持段式存储，而所有其他的机器都选择了页式存储；80x86对整型数据使用扩展的累加器，而其他的机器使用通用寄存器；80x86使用一个栈来保存浮点数据，而所有其他的机器在很久以前就废弃了执行栈。

尽管存在缺陷，80x86系统结构还是获得了巨大的成功。主要有三个方面原因：第一，被选为IBM PC的微处理器，使得在二进制上与80x86兼容变得非常重要；第二，摩尔定律保证了有足够的资源可使80x86在内部转化为RISC指令系统，然后执行类RISC指令。这两点使得它与大量PC软件二进制兼容，在性能上和RISC处理器不相上下；第三，大量的处理器出货使Intel有能力支付这种高代价的硬件转化。此外，大量的处理器出货使生产厂家可以跟得上学习曲线，这也在一定程度上降低了成本。

2.10谬误和易犯的错误

- **谬误：可以设计一个没有缺陷的系统结构。**

所有系统结构的设计都涉及一系列硬件和软件技术之间的折中。这些技术可能会随着时间的发展而发生变化，那些在设计者当初看来好像是正确的决定事后可能证明是错误的。

例如，在1975年，VAX的设计者们过分强调了代码大小的重要性，而没有估计到5年后译码的简化和流水线是那么重要。RISC系统中一个典型的例子就是延迟分支（见附录J）。对于５级流水线的处理器来说这容易解决，但如果处理器的流水线更长，在一个时钟周期内同时执行数条指令其困难就比较大。此外，几乎所有的系统结构最终都会产生地址空间不足的问题。

第二章　指令系统原理与实例

- 2.1 简介
- 2.2 指令集系统结构的分类
- 2.3 存储器寻址
- 2.4 操作数的类型
- 2.5 指令系统的操作
- 2.6 控制流指令
- 2.7 指令系统的编码
- 2.8 编译器的角色
- 2.9 MIPS系统结构
- 2.10 谬误和易犯的错误
- 2.11 结论

2.11 结论

- 20世纪60年代，栈系统结构流行。被认为能最好地配合高级语言——就当时的编译技术而言，这或许是对的。
- 20世纪70年代，设计的主要焦点是如何降低软件的费用，具体是用硬件来取代软件，或是提供能够简化软件设计者工作的高级结构。结果是：既产生了高级语言计算机系统结构，又出现了诸如VAX一类的强大系统结构。VAX具有大量的寻址方式、多种数据类型以及一个高度正交化的系统结构。
- 20世纪80年代，更成熟的编译技术以及重新强调机器效率的观点使更简单的系统又成为了主流，它主要基于RISC的load-store系统结构机器。

2.11 结论

- 20世纪90年代指令集系统结构发生了一些变化：
  - 地址空间翻倍：32位地址指令系统扩展到64位，寄存器的位数扩充到64位。附录C给出了3种从32位扩展到64位的系统结构的例子。
  - 通过条件执行实现条件分支转移的优化：条件分支严重地限制了性能的提升。因此用条件操作完成来代替条件分支有很大的好处，例如条件传送（参见附录G）已经被大多数指令系统采用。

2.11结论

- 20世纪90年代指令集系统结构发生了一些变化：

- **通过预取优化Cache性能**：Cache的一次不命中所花掉的时间与早期机器中一次缺页错误所花掉的时间一样多，因此，增加了预取指令来减少Cache不命中的代价。
- **对多媒体的支持**：大多数桌面和嵌入式指令系统都增加了对多媒体和DSP应用程序的支持。
- **更快的浮点操作**：附录I给出了一些提高浮点性能的新操作。如执行一个乘法和一个加法的操作，执行配对单精度的操作。