计算机系统结构

第1章 计算机系统结构的基础知识

目录

1.1 计算机系统结构的基本概念

1. 第一台通用电子计算机诞生于1946年
2. 计算机技术的飞速发展受益于两个方面
   - 计算机制造技术的发展
   - 计算机系统结构的创新
3. 经历了四个发展过程

时 间	原 因	每年的性能增长
1946年起的25年	两种因素都起着主要的作用	25%
20世纪70年代末～80年代初	大规模集成电路和微处理器出现，以集成电路为代表的制造技术的发展	约35%
80年代中开始	RISC结构的出现，系统结构不断更新和变革，制造技术不断发展	50%以上 维持了约16年
2003年以来	3个（见下页）	约20%

- 大功耗问题；
- 可以进一步有效地开发的指令级并行性已经很少；
- 存储器访问速度的提高缓慢。

系统结构的重大转折：

从单纯依靠指令级并行转向开发线程级并行和数据级并行。

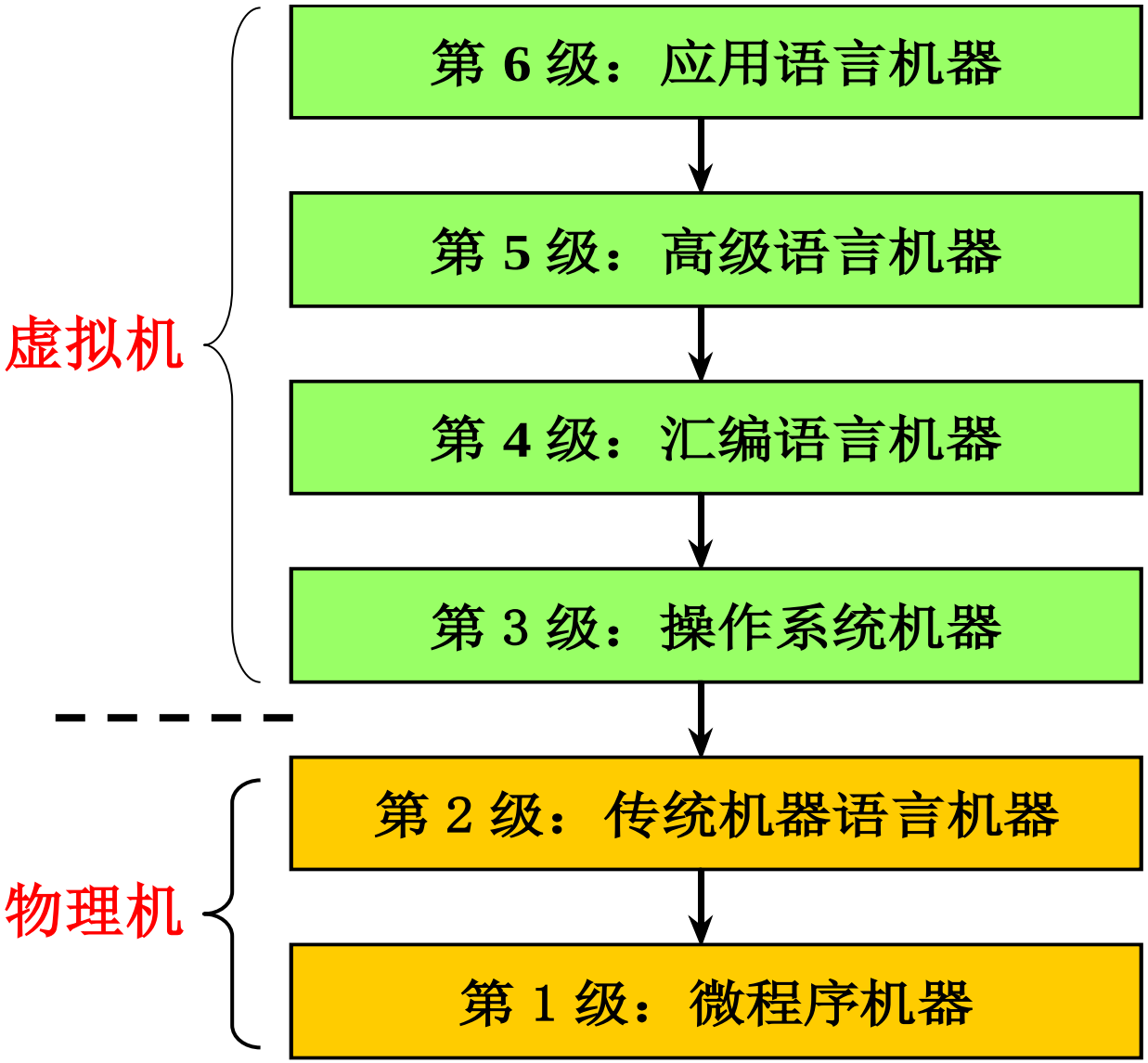
计算机系统结构在计算机的发展中有着极其重要的作用。

1.1.1 计算机系统的层次结构

1. 计算机系统＝硬件＋软件
2. 计算机语言从低级向高级发展

   高一级语言的语句相对于低一级语言来说功能更强，更便于应用，但又都以低级语言为基础。

3. 从计算机语言的角度，把计算机系统按功能划分成多级层次结构。
   - 每一层以一种语言为特征


第 6 级：应用语言机器
第 5 级：高级语言机器
第 4 级：汇编语言机器
第 3 级：操作系统机器
虚拟机
第 2 级：传统机器语言机器
第 1 级：微程序机器
物理机

- 物理机: 用硬件实现的机器(最下面的两级机器)
- 虚拟机：由软件实现的机器
- 各机器级的实现主要靠翻译或解释，或两者的结合。
  - **翻译（Translation）：**先用转换程序把高一级机器上的程序转换为低一级机器上等效的程序，然后再在这低一级机器上运行，实现程序的功能。
  - **解释（Interpretation）：**对于高一级机器上的程序中的每一条语句或指令，都是转去执行低一级机器上的一段等效程序。执行完后，再去高一级机器取下一条语句或指令，再进行解释执行，如此反复，直到解释执行完整个程序。

解释执行比翻译（编译）后再执行所花的时间多，但占用的存储空间较少。

1.1.2 计算机系统结构(Computer Architechture)的定义

1. 计算机系统结构的经典定义

传统机器程序员所看到的计算机属性，即概念性结构与功能特性。

（1964年 Amdahl在介绍IBM360系统时提出的）

2. 按照计算机系统的多级层次结构，不同级程序员所看到的计算机具有不同的属性。

3. 透明性(Transparency)

在计算机技术中，把这种本来存在的事物或属性，但从某种角度看又好像不存在的概念称为透明性。

底层机器的属性对于高层机器的程序员来说往往是透明的。

4. 广义的系统结构定义：指令系统结构、组成、硬件

（计算机设计的3个方面）

计算机系统结构的实质：

确定计算机系统中软硬件的界面，界面之上是软件实现的功能，界面之下是硬件和固件实现的功能。

1.1.3 计算机组成和计算机实现

1. 计算机系统结构（Computer Architecture）：计算机系统的软、硬件的界面

   即机器语言程序员所看到的传统机器级所具有的属性。

2. 计算机组成（Computer Organization）：计算机系统结构的逻辑实现

   - 包含物理机器级中的数据流和控制流的组成以及逻辑设计等。
   - 着眼于：物理机器级内各事件的排序方式与控制方式、各部件的功能以及各部件之间的联系。

1.1 计算机系统结构的基本概念

3. 计算机实现（Computer Implementation）：计算机组成的物理实现
   - 包括处理机、主存等部件的物理结构，器件的集成度和速度，模块、插件、底板的划分与连接，信号传输，电源、冷却及整机装配技术等。
   - 着眼于：器件技术（起主导作用）、微组装技术。

具有相同系统结构的计算机可以采用不同的计算机组成。

同一种计算机组成又可以采用多种不同的计算机实现。

乘法指令

指令系统是否有乘法指令---计算机系统结构的内容

乘法指令实现方式：专门的乘法器、加法器多步操作---计算机组成；

乘法器、加法器的物理实现，期间选定以及微组装技术---计算机实现。

主存容量与编址方式

计算机系统结构：主存容量与编址方式（按位、按字节或按字访问）。

计算机组成：主存速度多快、逻辑结构是否采用多体交叉。

计算机实现：主存系统的物理实现，如器件选定、逻辑电路的设计、微组装技术的使用。

4. 系列机

由同一厂家生产的具有相同系统结构、但具有不同组成和实现的一系列不同型号的机器。

例如 IBM公司的IBM370系列，Intel公司的x86系列等。

1.1.4 计算机系统结构的分类

常见的计算机系统结构分类法有3种：

Flynn分类法、冯氏分类法和Handler分类法

1. Flynn分类法 （M.J.Flynn 1966年提出）

- 按照指令流和数据流的多倍性进行分类。
  - 指令流：计算机执行的指令序列
  - 数据流：由指令流调用的数据序列
  - 多倍性：在系统最受限的部件上，同时处于同一执行阶段的指令或数据的最大数目。

- 把计算机系统的结构分为4类
  - 单指令流单数据流SISD

（Single Instruction stream Single Data stream）
  - 单指令流多数据流SIMD

（Single Instruction stream Multiple Data stream）
  - 多指令流单数据流MISD

（Multiple Instruction stream Single Data stream）
  - 多指令流多数据流MIMD

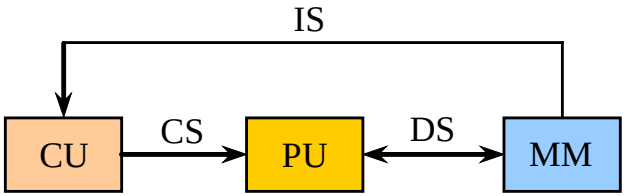
（Multiple Instruction stream Multiple Data stream）

➢ 4类计算机的基本结构

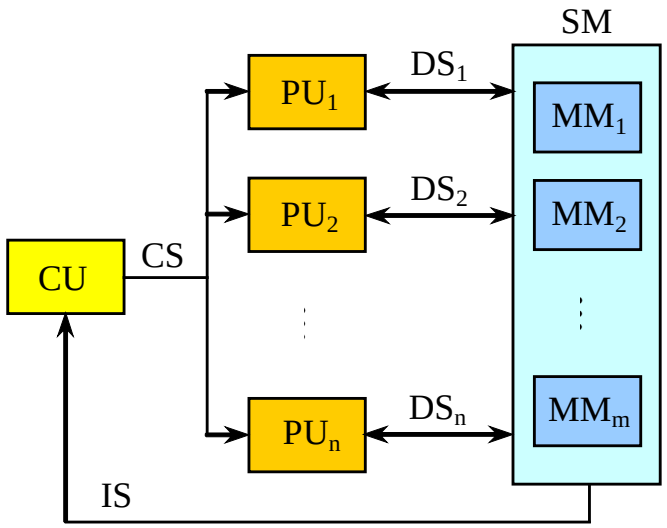
IS：指令流　　CU：控制部件

DS：数据流　　PU：处理部件

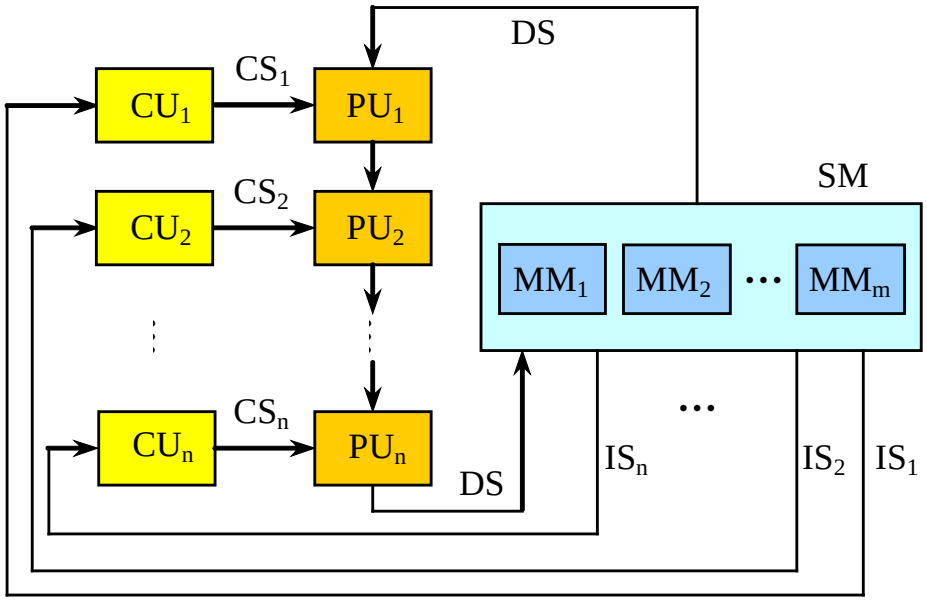
CS：控制流　　MM和SM：存储器



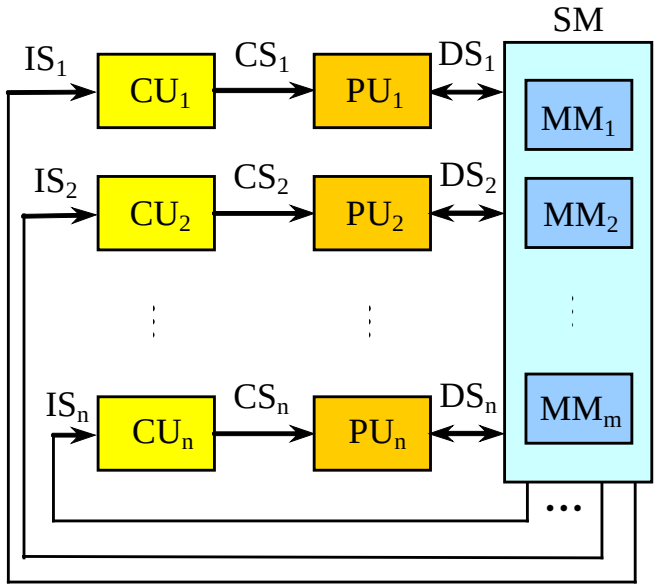
（a）SISD 计算机



（b）SIMD 计算机



（c）MISD 计算机

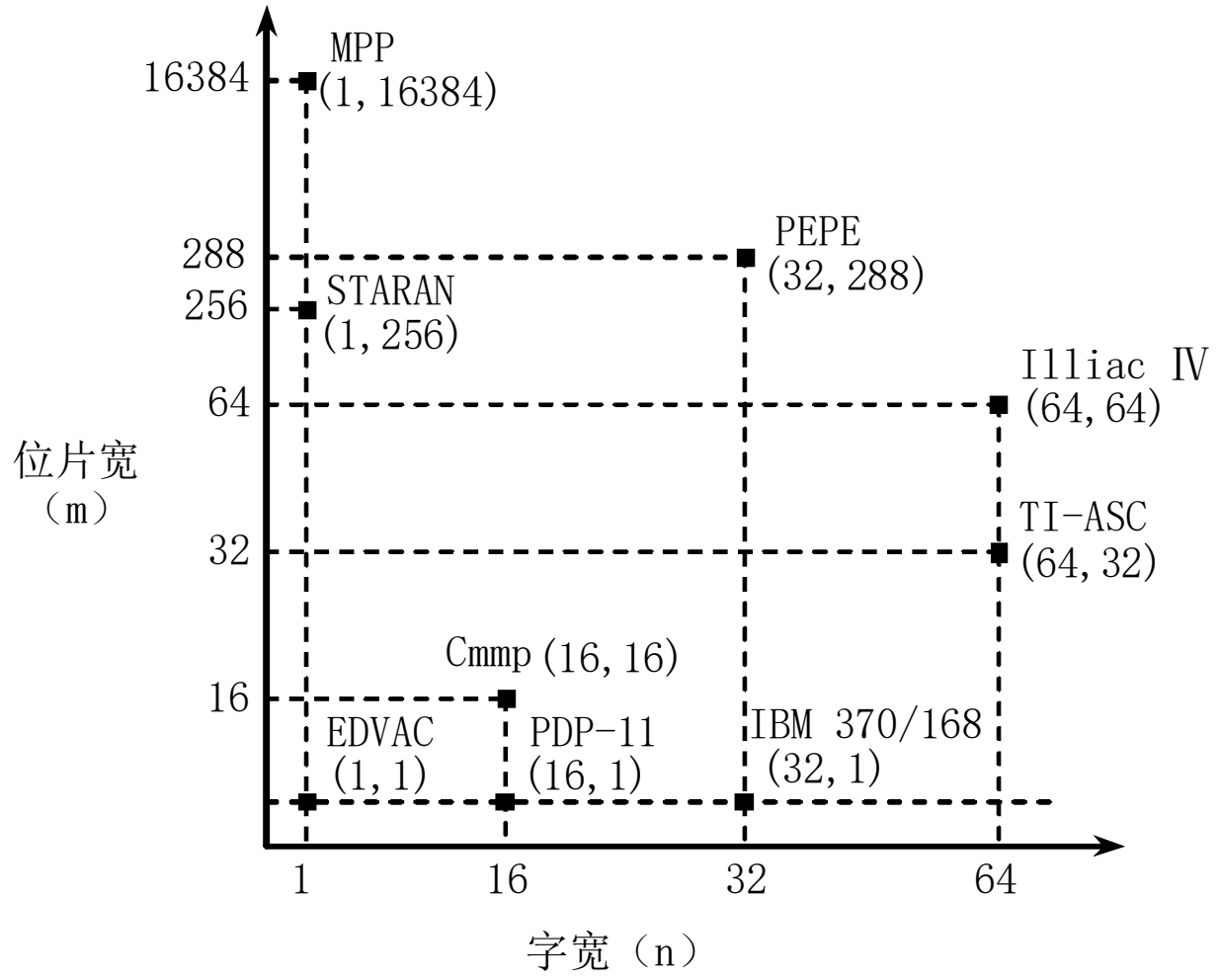


（d）MIMD 计算机

2. 冯氏分类法 （美籍华人冯泽云先生1972年提出）

- 用系统的最大并行度对计算机进行分类。
- 最大并行度：计算机系统在单位时间内能够处理的最大的二进制位数。

用平面直角坐标系中的一个点代表一个计算机系统，其横坐标表示字宽（n位），纵坐标表示一次能同时处理的字数（m字）。m×n就表示了其最大并行度。


MPP
(1,16384)
16384
288
256
64
32
16
位片宽
（m）
PEPE
(32,288)
STARAN
(1,256)
Illiac Ⅳ
(64,64)
TI-ASC
(64,32)
Cmmp (16,16)
EDVAC
(1,1)
PDP-11
(16,1)
IBM 370/168
(32,1)
1
16
32
64
字宽（n）

- 4类不同最大并行度（P_m）的计算机系统结构
  - 字串位串：n＝1，m＝1。
    (第一代计算机发展初期的纯串行计算机)
  - 字串位并： n＞1，m＝1。这是传统的单处理机，同时处理单个字的多个位，如16位、32位等。
  - 字并位串：n＝1，m＞1。同时处理多个字的同一位（位片）。
  - 字并位并：n＞1，m＞1。同时处理多个字的多个位。
- 平均并行度 （P_a）

  与最大并行度密切相关的一个指标。

  取决于系统的运用程度，与应用程序有关。

假设每个时钟周期内能同时处理的二进制位数为P_i，则T个时钟周期内的平均并行度为：

$$P_a = \frac{\sum_{i=1}^{T} P_i}{T}$$

系统在T个时钟周期内的平均利用率定义为：

$$\mu = \frac{P_a}{P_m} = \frac{\sum_{i=1}^{T} P_i}{TP_m}$$

3. Handler分类法（德国Wolfgan Handler 1977年提出）

- 根据并行度和流水线对计算机进行分类。
- 把计算机的硬件结构分成3个层次
  - 程序控制部件（**PCU**）的个数**k**
  - 算术逻辑部件（**ALU**）或处理部件（**PE**）的个数**d**
  - 每个算术逻辑部件包含基本逻辑线路(**ELC**)的套数**w**
- 用公式表示

  t（系统型号）＝（k，d，w）
- 进一步改进

  t（系统型号）＝（k×k’，d×d’，w×w’）
  - **k’**：宏流水线中程序控制部件的个数
  - **d’**：指令流水线中算术逻辑部件的个数
  - **w’**：操作流水线中基本逻辑线路的套数

- 例如：Cray-1有1个CPU，12个相当于ALU或PE的处理部件，可以

最多实现8级流水线。字长为64位，可以实现1～14位流水线处理。所以Cray-1系统结构可表示为：

t（Cray-1）＝(1，12×8，64×（1～14）)

几个例子：

t（PDP-11）＝(1，1，16)

t（Illiac IV）＝(1，64，64)

t（STARAN）＝(1，8192，1)

t（Cmmp）＝(16，1，16)

t（PEPE）＝(1×3，288，32)

t（TI-ASC）＝(1，4，64×8)

1.2 计算机系统的设计

1.2.1 计算机系统设计的定量原理

4个定量原理：

1. 以经常性事件为重点
   - 对经常发生的情况采用优化方法的原则进行选择，以得到更多的总体上的改进。
   - 优化是指分配更多的资源、达到更高的性能或者分配更多的电能等。

2. Amdahl定律

加快某部件执行速度所能获得的系统性能加速比，受限于该部件的执行时间占系统中总执行时间的百分比。

系统性能加速比：

$$\text{加速比} = \frac{\text{系统性能}_{\text{改进后}}}{\text{系统性能}_{\text{改进前}}} = \frac{\text{总执行时间}_{\text{改进前}}}{\text{总执行时间}_{\text{改进后}}}$$

- 加速比依赖于两个因素
  - 可改进比例（F_e）：在改进前的系统中，可改进部分的执行时间在总的执行时间中所占的比例。

    它总是小于等于1。

    例如：一个需运行60秒的程序中有20秒的运算可以加速，

    那么这个比例就是20/60。
  - 部件加速比（S_e）：可改进部分改进以后性能提高的倍数。它是改进前所需的执行时间与改进后执行时间的比。

    一般情况下部件加速比是大于1的。

    例如：若系统改进后，可改进部分的执行时间是2秒，

    而改进前其执行时间为5秒，则部件加速比为5/2。

- 改进后程序的总执行时间T_n

$$T_n = T_0\left(1 - Fe + \frac{Fe}{Se}\right)$$

  - T_0：改进前整个程序的执行时间
  - $1-F_e$：不可改进比例

系统加速比S_n为改进前与改进后总执行时间之比:

$$S_n = \frac{T_0}{T_n} = \frac{1}{(1 - Fe) + \frac{Fe}{Se}}$$

理论上，4核处理器的指令处理速度是单核处理器的4倍。
但是只有10%的指令是可以在4个核心上并行处理的。
答：Fe=0.1，　　Se=4，　　（1-Fe）=0.9

$$T_n = T_o(1 - F_e + \frac{F_e}{S_e}) = T_o(1 - 0.1 + \frac{0.1}{4}) = 0.925T_o$$

$$S_n = \frac{T_o}{T_n} = \frac{1}{0.925} = 1.081$$

例1.1 将计算机系统中某一功能的处理速度加快15倍，但该功能的处理时间仅占整个系统运行时间的40%，则采用此增强功能方法后，能使整个系统的性能提高多少？

解 由题可知： $F_e = 40\% = 0.4$

$S_e = 15$

根据Amdahl定律可知：

$$S_n = \frac{1}{(1-Fe)+\frac{Fe}{Se}} = \frac{1}{(1-0.4)+\frac{0.4}{15}} \approx 1.6$$

采用此增强功能方法后，能使整个系统的性能提高到原来的1.6倍。

例1.2 某计算机系统采用浮点运算部件后，使浮点运算速度提高到原来的25倍，而系统运行某一程序的整体性能提高到原来的4倍，试计算该程序中浮点操作所占的比例。

解 由题可知： $S_e = 25$ $S_n = 4$

根据Amdahl定律可知：

$$4 = \frac{1}{(1 - Fe) + \frac{Fe}{25}}$$

由此可得：Fe = 78.1%

即程序中浮点操作所占的比例为78.1%。

1.2 计算机系统的设计


当$F_e \to 0$, $Sn \to 1$
S_n
$\frac{1}{1-Fe}$
1
0
0.0
1
S_e
由图中曲线可知，随着Se的增大，Sn的增速越来越慢，且收敛到$\frac{1}{1-Fe}$
Amdahl定律的性能递减规则：
如果仅仅对计算机中的一部分做性能改进，则改进越多，所得到的总体性能的提升就越有限。

- Amdahl定律：一种性能改进的递减规则

$$S_n = \frac{T_0}{T_n} = \frac{1}{(1-Fe) + \frac{Fe}{Se}}$$

如果仅仅对计算任务中的一部分做性能改进，则改进得越多，所得到的总体性能的提升就越有限。

$$S_e \to \infty$$

$$S_n = \frac{T_0}{T_n} = \frac{1}{(1-Fe)}$$

- 重要推论：如果只针对整个任务的一部分进行改进和优化，那么所获得的加速比不超过：

1/（1－可改进比例）

3. CPU性能公式

- 执行一个程序所需的CPU时间

CPU时间 = 执行程序所需的时钟周期数 × 时钟周期时间

其中：时钟周期时间是系统时钟频率的倒数。

- 每条指令执行的平均时钟周期数CPI

（Cycles Per Instruction）

CPI = 执行程序所需的时钟周期数 / IC

IC：所执行的指令条数

- 程序执行的CPU时间可以写成

CPU时间 = IC × CPI × 时钟周期时间

- CPU的性能取决于三个参数
  - 时钟周期时间：取决于硬件实现技术和计算机组成；
  - CPI：取决于计算机组成和指令系统的结构；
  - IC：取决于指令系统的结构和编译技术。
- 对CPU性能公式进行进一步细化

假设：计算机系统有n种指令；

CPI_i ：第i种指令的处理时间；

IC_i ：在程序中第i种指令出现的次数；

则：

$$\text{CPU时钟周期数} = \sum_{i=1}^{n} (CPI_i \times IC_i)$$

CPU时间 = 执行程序所需的时钟周期数 × 时钟周期时间

$$= \sum_{i=1}^{n} (CPI_i \times IC_i) \times \text{时钟周期时间}$$

CPI可以表示为：

$$CPI = \frac{\text{时钟周期数}}{IC} = \frac{\sum_{i=1}^{n} (CPI_i \times IC_i)}{IC} = \sum_{i=1}^{n} (CPI_i \times \frac{IC_i}{IC})$$

其中：(IC_i/IC)反映了第i种指令在程序中所占的比例。

例1.3 假设FP指令的比例为25%，其中，FPSQR占全部指令的比例为2%，FP操作的CPI为4，FPSQR操作的CPI为20，其他指令的平均CPI为1.33。现有两种改进方案，第一种是把FPSQR操作的CPI减至2，第二种是把所有的FP操作的CPI减至2，试比较两种方案对系统性能的提高程度。

解 没有改进之前，每条指令的平均时钟周期CPI为：

$$CPI = \sum_{i=1}^{n}\left(CPI_i \times \frac{IC_i}{IC}\right) = (4 \times 25\%) + (1.33 \times 75\%) = 2$$

（1）采用第一种方案

FPSQR操作的CPI由$CPI_{FPSQR}=20$减至$CPI'_{FPSQR}=2$，则整个系统的指令平均时钟周期数为：

$$CPI_1=CPI—（CPI_{FPSQR}—CPI'_{FPSQR}）\times 2\%$$

$$=2—(20—2)\times 2\%=1.64$$

（2）采用第二种方案

所有FP操作的CPI由$CPI_{FP}=4$减至$CPI'_{FP}=2$，则整个系统的指令平均时钟周期数为：

$$CPI_2=CPI—（CPI_{FP}—CPI'_{FP}）\times 25\%$$

$$=2—(4—2)\times 25\%=1.5$$

从降低整个系统的指令平均时钟周期数的程度来看，第二种方案优于第一种方案。

例1.4 考虑条件分支指令的两种不同设计方法：

（1）CPU_1：通过比较指令设置条件码，然后测试条件码进行分支。

（2）CPU_2：在分支指令中包括比较过程。

在这两种CPU中，条件分支指令都占用2个时钟周期，而所有其它指令占用1个时钟周期。对于CPU_1，执行的指令中分支指令占30%；由于每条分支指令之前都需要有比较指令，因此比较指令也占30%。由于CPU_1在分支时不需要比较，因此CPU_2的时钟周期时间是CPU_1的1.35倍。问：哪一个CPU更快？如果CPU_2的时钟周期时间只是CPU_1的1.15倍，哪一个CPU更快呢？

解 我们不考虑所有系统问题，所以可用CPU性能公式。占用2个时钟周期的分支指令占总指令的30%，剩下的指令占用1个时钟周期。所以

$$CPI_1 = 0.3 \times 2 + 0.70 \times 1 = 1.3$$

则CPU_1性能为：

$$总CPU时间_1 = IC_1 \times 1.3 \times 时钟周期_1$$

根据假设，有：

$$时钟周期_2 = 1.35 \times 时钟周期_1$$

在CPU_2中没有独立的比较指令，所以CPU_2的程序量为CPU_1的70%，分支指令的比例为：

$$30\%/70\% = 42.8\%$$

这些分支指令占用2个时钟周期，而剩下的57.2%的指令占用1个时钟周期，因此：

$$CPI_2 = 0.428 \times 2 + 0.572 \times 1 = 1.428$$

因为CPU_2不执行比较，故：

$$IC_2 = 0.7 \times IC_1$$

因此CPU_2性能为：

$$\text{总CPU时间}_2 = IC_2 \times CPI_2 \times \text{时钟周期}_2$$

$$= 0.7 \times IC_1 \times 1.428 \times (1.35 \times \text{时钟周期}_1)$$

$$= 1.349 \times IC_1 \times \text{时钟周期}_1$$

在这些假设之下，尽管CPU_2执行指令条数较少，CPU_1因为有着更短的时钟周期，所以比CPU_2快。

如果CPU_2的时钟周期时间仅仅是CPU_1的1.15倍，则

$$时钟周期_2 = 1.15 \times 时钟周期_1$$

CPU_2的性能为：

$$总CPU时间_2 = IC_2 \times CPI_2 \times 时钟周期_2$$

$$= 0.7 \times IC_1 \times 1.428 \times (1.15 \times 时钟周期_1)$$

$$= 1.15 \times IC_1 \times 时钟周期_1$$

因此CPU_2由于执行更少指令条数，比CPU_1运行更快。

4. 程序的局部性原理

程序执行时所访问的存储器地址分布不是随机的，而是相对地簇聚。

- 常用的一个经验规则

  程序执行时间的90%都是在执行程序中10%的代码。

- 程序的时间局部性

  程序即将用到的信息很可能就是目前正在使用的信息。

- 程序的空间局部性

  程序即将用到的信息很可能与目前正在使用的信息在空间上相邻或者临近。

1.2.2 计算机系统设计者的主要任务

1. 计算机系统设计者的任务包括：指令系统的设计、数据表示的设计、功能的组织、逻辑设计以及其物理实现等。

2. 设计一个计算机系统大致要完成3个方面的工作。
   - 确定用户对计算机系统的功能、价格和性能的要求
     - 计算机系统设计者的目标

       设计出能满足用户的功能需求、有较长的生命周期、且又具有很高的性能价格比的系统。
     - 功能需求：根据市场的需要以及所设计系统的应用领域来确定

1.2 计算机系统的设计

- 应用领域

  专用还是通用？面向科学计算还是面向商用处理？

- 软件兼容

  软件兼容是指一台计算机上的程序不加修改就可以搬到另一台计算机上正常运行。

- 操作系统需求

  包括地址空间大小、存储管理、保护等。从系统结构上对操作系统的需求提供支持，是很重要的一点。

- 标准

  确定系统中哪些方面要采用标准以及采用什么标准。

  如：浮点数标准、I/O总线标准、网络标准、程序设计语言标准等。

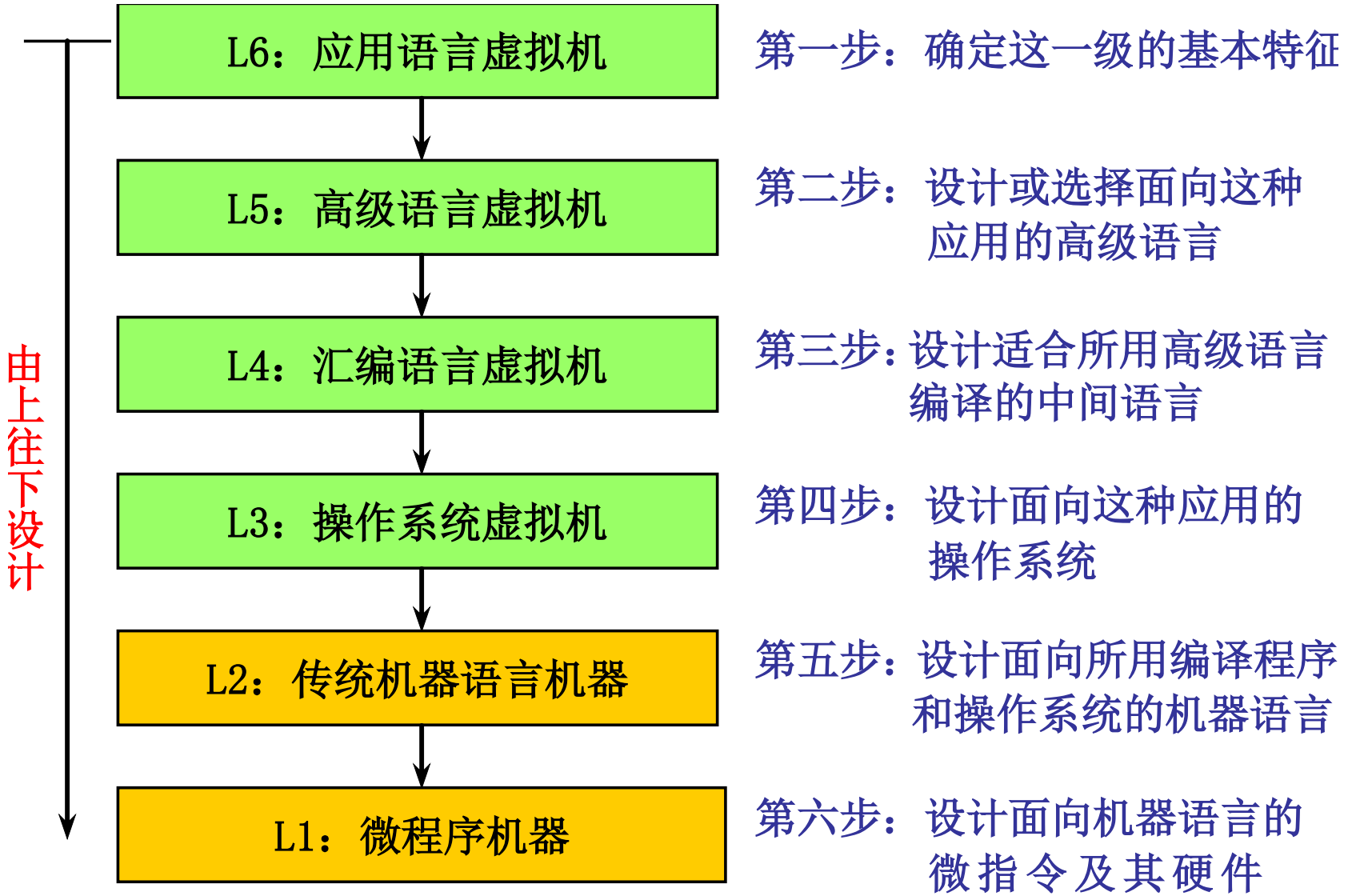
- 软硬件功能分配
  - 考虑如何优化设计？

    必须考虑软硬件功能的合理分配。
  - 软件和硬件在实现功能上是等价的
    - 用软件实现的优点：设计容易、修改简单，而且可以减少硬件成本。但是所实现的功能的速度较慢。
    - 用硬件实现的优点：速度快、性能高，但它修改困难，灵活性差。
  - 在软硬件之间进行折中和取舍。

- 设计出生命周期长的系统结构
  - 特别注意计算机应用和计算机技术的发展趋势
  - 设计出具有一定前瞻性的系统结构，以使得它具有较长的生命周期。

1.2.3 计算机系统设计的主要方法

1. “由上往下”（top-down）设计
   - 从层次结构中的最上面一级开始，逐层往下设计各层的机器。


L6：应用语言虚拟机
L5：高级语言虚拟机
L4：汇编语言虚拟机
L3：操作系统虚拟机
L2：传统机器语言机器
L1：微程序机器
由上往下设计
第一步：确定这一级的基本特征
第二步：设计或选择面向这种应用的高级语言
第三步：设计适合所用高级语言编译的中间语言
第四步：设计面向这种应用的操作系统
第五步：设计面向所用编译程序和操作系统的机器语言
第六步：设计面向机器语言的微指令及其硬件

- 首先确定面对使用者的那级机器的基本特征、数据类型和格式、基本命令等。
- 然后再逐级往下设计，每级都考虑如何优化上一级的实现。

➢ 适合于专用机的设计，而不适合通用机的设计。

2. “由下往上”（bottom-up）设计

➢ 从层次结构的最下面一级开始，逐层往上设计各层的机器。

➢ 采用这种方法时，软件技术完全处于被动状态，这会造成软件和硬件的脱节，使整个系统的效率降低。

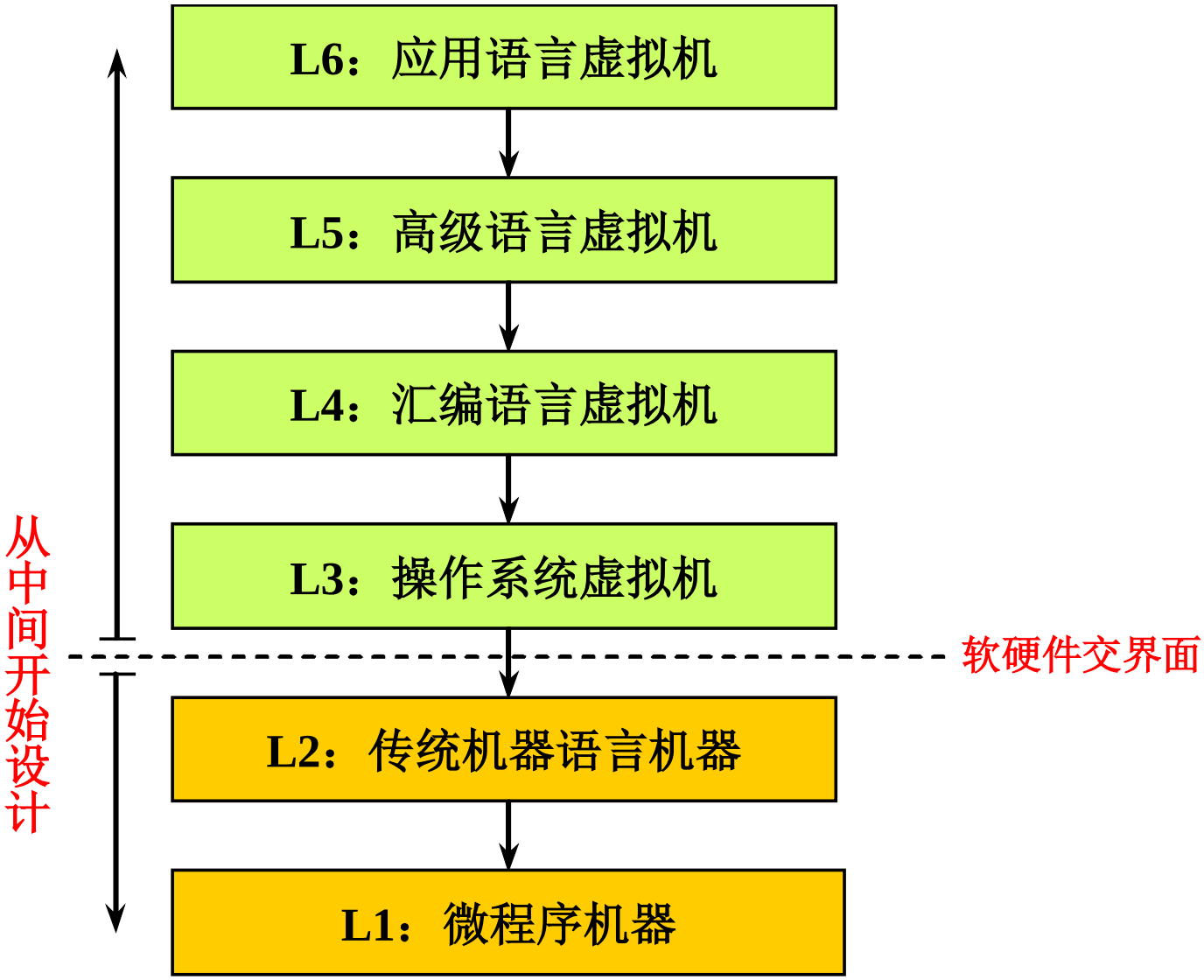
（在早期被采用得比较多，现在已经很少被采用了）

3. “从中间开始”（middle-out）设计

- “由上往下”和“由下往上”设计方法的主要缺点

  软、硬件设计分离和脱节

- 解决方法：综合考虑软、硬件的分工，从中间开始设计。
  - “中间”：层次结构中的软硬件的交界面，目前一般是在传统机器语言机器级与操作系统机器级之间。
- 从中间开始设计
  - 首先要进行软、硬件功能分配，确定好这个界面。
  - 然后从这个界面开始软件设计者开始往上设计操作系统、汇编、编译系统等，硬件设计者开始往下设计传统机器级、微程序机器级等。


L6：应用语言虚拟机
L5：高级语言虚拟机
L4：汇编语言虚拟机
L3：操作系统虚拟机
软硬件交界面
L2：传统机器语言机器
L1：微程序机器
从中间开始设计