计算机体系结构基础

胡伟武、苏孟豪

第05章：计算机组成原理和结构

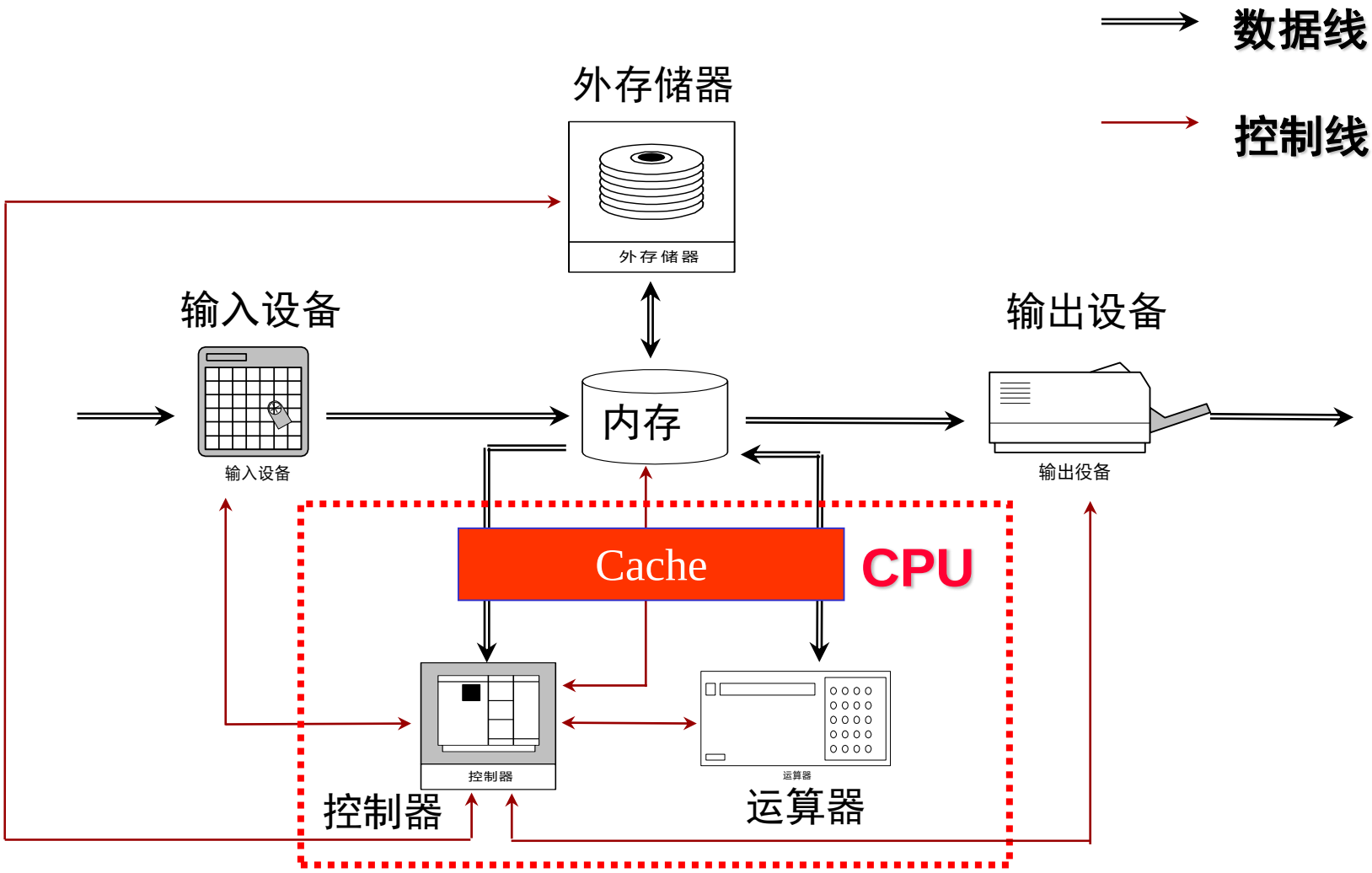
- 冯诺依曼结构
- 计算机系统主要组成部件
- 计算机硬件结构的演进
- 处理器和IO间通信

冯诺依曼结构

冯诺依曼结构基本原理

- 存储程序和指令驱动执行
  - ①计算机由存储器、运算器、控制器、输入设备、输出设备五部分组成，其中运算器和控制器合称为中央处理器（Central Processing Processor，简称CPU）或处理器。**计算机从输入设备接收程序和数据，存放在存储器中；CPU运行程序处理数据；最后将结果数据通过输出设备输出。**
  - ②存储器是按地址访问的线性编址的一维结构，每个单元的位数固定。指令和数据不加区别混合存储在同一个存储器中。
  - ③控制器从存储器中取出指令并根据指令要求发出控制信号控制计算机的操作。控制器中的程序计数器指明要执行的指令所在的存储单元地址。程序计数器一般按顺序递增，但可按指令要求而改变。
  - ④以运算器为中心，输入输出（Input/Output，简称IO）设备与存储器之间的数据传送都经过运算器。

计算机硬件系统的组成


数据线
控制线
外存储器
外存储器
输入设备
输入设备
内存
输出设备
输出设备
Cache
CPU
控制器
控制器
运算器
运算器

冯诺依曼结构的演进

- 以下演进没有改变存储程序和指令驱动执行的特点
  - ①以运算器为中心改进为以存储器为中心，数据流向更加合理，从而使运算器、存储器和IO设备能够并行工作。
  - ②由单一的集中控制改进为分散控制。早期的计算机工作速度低，运算器、存储器、控制器和IO设备可以在同一个时钟信号的控制下同步工作。现在运算器、存储器与IO设备的速度差异很大，需要异步分散控制。
  - ③从基于串行算法改进为适应并行算法，出现了流水线处理器、超标量处理器、向量处理器、多核处理器、对称多处理机（Symmetric Multiprocessor，简称SMP）、大规模并行处理机（Massively Parallel Processing，简称MPP）和机群系统等。
  - ④出现了为适应特殊需要的专用计算机，如图形处理器（Graphic Processing Unit，简称GPU）、数字信号处理器（Digital Signal Processor，简称DSP）等

冯诺依曼结构的优缺点

- 本质特征
  - 存储程序和指令驱动执行
  - 目前的计算机没有能突破该特征的，都是对冯诺依曼结构的变种（如哈佛结构、并行结构等）
- 优点
  - 自动、快速执行
- 缺点：喂不饱的运算器
  - 指令驱动的顺序执行
  - CPU和存储器分开，而且越来越远

计算机的组成部件

计算机组成部分

- 计算机五大部分逻辑组成
  - 输入设备：键盘、鼠标、扫描仪……
  - 输出设备：显示器、打印机、扬声器……
  - 运算器：加、减、乘、除、与、或、非……
  - 控制器：取指、译码、跳转、访存……
  - 存储器：内存、硬盘、SSD……
- 物理上分为三大部分
  - CPU：运算器+控制器+高速缓存（内存的一个子集）
  - 内存：主要由DRAM组成
  - IO设备：I和O的控制和通信是一致的
  - 三大部分之间的通信和同步对性能至关重要

运算器（1）

- 运算器是计算机中执行（算术和逻辑）运算指令的部件
  - 算术运算：加、减、乘、除等
  - 逻辑运算：与、或、非等
- 运算器的组成
  - 运算器包括算术和逻辑运算部件、移位部件、寄存器等。复杂运算如乘除法、开方及浮点运算可用程序实现或由运算器实现。寄存器既可用于保存数据，也可用于保存地址。运算器还可设置条件码寄存器等专用寄存器，条件码寄存器保存当前运算结果的状态，如运算结果是正数、负数或零，是否溢出等。

运算器（2）

- 运算器支持的运算类型经历了从简单到复杂的过程
  - 最初只有简单的定点加减和基本逻辑运算，复杂运算如乘除通过加减、移位指令构成的数学库完成；后来逐渐出现硬件定点乘法器和除法器
  - 浮点运算器以协处理器的形式出现在计算机中（**Intel 8087**协处理器），包含二进制浮点数的加、减、乘、除等运算
  - 上世纪**90**年代开始，微处理器中出现的**SIMD**向量运算器
  - 部分处理器实现了超越函数硬件运算单元，如**sin**、**cos**、**exp**、**log**等
  - 部分用于银行业务处理的计算机（**IBM POWER**系列）还实现了十进制定、浮点数的运算器

运算器（3）

- 随着晶体管集成度的不断提升，处理器中所集成的运算器的数量也持续增加，通常将具有相近属性的一类运算组织在一起构成一个运算单元。
  - 不同的处理器有不同的运算单元组织，有的倾向于每个单元大而全，有的倾向于每个单元的功能相对单一
  - 处理器中包含的运算单元数目也逐渐增加，从早期的单个运算单元逐渐增加到多个运算单元，运算单元的个数主要受限于寄存器堆读写端口个数
  - 运算单元一般按照定点、浮点、访存、向量等大类来组织，也有混合的，如SIMD部件既能做定点、也能做浮点运算，定点部件也可以做访存地址计算

运算器（4）

- 经典商用处理器运算单元数目举例

Alpha21264	MIPS R10000	HP PA-8700	Ultra SPARC-III	POWER4
arith./logic unit shift unit mult unit add/logic unit shift unit MVI/PLZ unit arith./logic unit arith./logic unit FP add unit FP mult unit FP div/sqrt unit	arith./logic unit shift unit arith./logic unit mult/div unit FP add/sub unit FP comp unit FP conversion unit FP mult unit FP div/sqrt unit	2 arith./logic units 2 shift merge units 2 FP MAC units 2 FP div/sqrt units	2 arith units logic unit shift unit FP adder unit graphic unit FP div/sqrt unit FP mult unit graphic unit	2 fixed-point units 2 floating-point units

- 近期商用处理器运算单元数目举例

Intel Ivybridge	AMD Bulldozer	POWER7	Loongson GS464E
ALU/LEA/Shift/128b MUL/128b Shift/256b FMUL/256b Blend; ALU/LEA/Shift/128b ALU /128bit Shuffle/256b FADD; ALU/Shift/Branch/128b ALU/128b Shuffle/256b Shuffle/256b Blend	ALU/IMUL ALU/IDIV/Count 128b FMAC/IMAC 128b FMAC/XBAR 128b MMX 128b MMX/FSTO	2 fixed-point units 2 floating-points/vector units	2 fixed-point/DSP units 2 floating-points/vector units

控制器（1）

- 控制器控制指令流和每条指令的执行，内含程序计数器和指令寄存器等。
  - 程序计数器存放当前执行指令的地址，指令寄存器存放当前正在执行的指令。控制器还产生一定频率的时钟脉冲，用于计算机各组成部分的同步。
  - 指令通过译码产生控制信号，用于控制运算器、存储器、IO设备的工作。这些控制信号可以用硬连线逻辑产生，也可以用微程序产生，也可以两者结合产生。
  - 为了获得高指令吞吐率，可以采用指令重叠执行的流水线技术，以及同时执行多条指令的超标量技术。当遇到执行时间较长或条件不具备的指令时，把条件具备的后续指令提前执行（称为乱序执行）可以提高流水线效率。

控制器（2）

- 控制器的功能
  - 计算机执行指令一般包含以下过程：从存储器取指令并对取回的指令进行译码，从存储器或寄存器读取指令执行需要的操作数，执行指令，把执行结果写回存储器或寄存器。上述过程称为一个指令周期。计算机不断重复指令周期直到完成程序的执行。上述过程除了执行指令由运算器完成，其余过程由控制器完成。主要功能包括：从存储器/IO读取指令、解析指令（译码）、从/向存储器/IO中读取/写入操作数据、根据译码内容控制运算器进行相应操作。
- 控制器的性能
  - 体系结构研究的一个永恒主题就是不断加速上述指令执行周期，从而提高计算机运行程序的效率。
  - 晶体管集成度提升后，计算机中运算器资源不再是性能瓶颈，而给运算器及时供上指令和数据则成为影响性能的首要因素。

控制器提升性能技术

- 通过控制器设计以提升处理器性能的技术主要包括：
  - 指令流水线技术（时间重叠）
  - 乱序执行技术（可超车）
  - 超标量技术（多车道）
  - 转移预测技术（供指令）
  - 预取及高速缓存等（供数据，也可以作为存储器的一部分）

指令流水线技术

- 指令流水线技术将一条指令的执行拆分为多个阶段（如分为取指、译码、执行、访存、写回阶段），从而减少每个时钟周期的工作量以提升主频；并允许多条指令的不同阶段重叠执行实现并行处理。
- 通过单位时间内执行的指令数目增加来提升性能。

RISC技术提高指令流水线效率

- **RISC指令采用load-store结构，运算指令只访问寄存器，访存指令负责寄存器和内存/IO的数据交换。**
  - 运算器只需比较指令的寄存器号来判断指令间的数据相关，从而支持高效的流水线、多发射及乱序执行技术。**RISC结构大大简化了指令间的关系，便于指令调度。**
  - **X86系列从Pentium III开始，把CISC指令翻译成若干RISC微操作以提高指令流水线效率，如Haswell微结构最多允许192个内部微操作乱序执行。**
- **自从1940年代发明电子计算机以来，处理器结构和指令系统经历了一个由简单到复杂，由复杂到简单，又由简单到复杂的否定之否定过程。**
  - 早期处理器结构受工艺技术的限制，不可能做得很复杂。
  - **1960年代后流水线技术、动态调度技术、向量机技术被广泛使用。**
  - **1980年代提出的RISC技术简化了结构，有利于高效实现**
  - 后来随着深度流水、超标量、乱序执行的实现，**RISC结构变得越来越复杂**

乱序执行技术

- 通过乱序执行缓解由于指令相关引起的阻塞
  - 三类指令相关：数据相关（RAW、WAW、WAR）、控制相关、结构相关
  - 指令动态调度允许长延迟指令（如除法或cache不命中的访存）后面的源操作数准备好的指令越过长延迟指令执行，从而提高流水线效率
  - 乱序执行时，指令在译码后的读寄存器阶段，判断指令需要的操作数有没有准备好。如果操作数已经准备好，就进入执行阶段；否则就进入称为保留站或者发射队列的队列中等待，直到操作数准备好后再进入执行阶段。
  - 为保证执行结果符合程序规定的要求，乱序执行的指令要有序结束。为此，指令乱序执行前按程序的顺序进入重排序缓冲（ROB），指令执行后的结果写入称为重命名寄存器的临时寄存器，指令只能按照其在ROB中的次序逐条将存放在重命名寄存器的结果提交到目标寄存器或存储器中。
  - 通过把结构寄存器的访问重定向到重命名寄存器，两组执行不同运算但使用同一结构寄存器的指令可以使用不同的重命名寄存器，从而避免多条指令访问同一个结构寄存器时该寄存器成为串行化瓶颈

乱序执行技术举例

- **CDC6600使用计分板最早实现了指令的乱序执行，但是写后读相关和写后写相关引起的冲突仍需要通过阻塞流水线避免。**
- **IBM 360/91采用Tamosulo算法实现指令乱序执行，通过寄存器重命名技术解决写后读相关和写后写相关，不再需要阻塞流水线。**
- **现代处理器大多采用乱序执行技术，其基本设计原理基本脱胎于Tamosulo算法，不过它们在发射队列、重命名寄存器和ROB的具体结构和容量上存在存在区别。**

	Intel Ivybridge	AMD Bulldozer	IBM POWER7	Loongson GS464E
发射队列	54-项 统一	60-项 浮点(双核共享) 40-项 定点/访存	48-项 统一	32-项 浮点 32-项 定点 32-项 访存
重命名寄存器	160 项定点 144 项浮点	96 项定点 160 项浮点(双核共享)	80 项定点/浮点; 56 项CR; 40 项XER, 24 项Link&Count	128 项定点 128 项浮点 16 项Acc 32 项DSPCtrl 32 项FCR
ROB	168项	128项	120项	128项

超标量技术

- 超标量技术允许指令流水线的每一阶段同时处理多条指令，进一步提高单位时间内执行的指令数目，从而提升性能。
  - **Alpha21264：取指4条，发射6条，写回6条，提交11条**
- 超标量结构使得寄存器端口、保留站端口、ROB端口和功能部件数目都需要增加。
  - **Alpha21264：寄存器堆8个读端口6个写端口、2个访存部件**
- 超标量结构在指令译码和重命名时不仅要判断前后拍指令的数据相关，还需要判断同一拍中多条指令间的数据相关。
- 超标量流水线的结构复杂度与发射宽度成平方关系，因此不可能做得很大。

转移预测技术

- 冯·诺依曼结构指令驱动的特点使转移指令成为提高流水线效率的瓶颈。
  - 转移指令的后续指令需要等待转移指令执行结果确定后才能取指，导致转移指令和后续指令之间不能重叠执行，降低了流水线效率。
  - 典型程序平均每5~10条指令就有一条转移指令，现代处理器流水线普遍在10~20级之间，由于转移指令引起的流水线阻塞成为提高流水线效率的重要瓶颈。
- 采用转移预测技术消除转移指令引起的指令流水线阻塞。
  - 转移预测技术根据当或其他转移指令的历史行为，在转移指令的取指或译码阶段就预测该转移指令的跳转方向和目标地址，并据此进行后续指令的取指。
  - 转移指令执行后，根据已确定的跳转方向和跳转目标地址对预测结果进行判定。如果发生转移预测错误，则需要取消流水线中的后续指令。
  - 常见预测技术有分支历史表BHT、分支目标缓冲BTB、返回地址栈RAS等

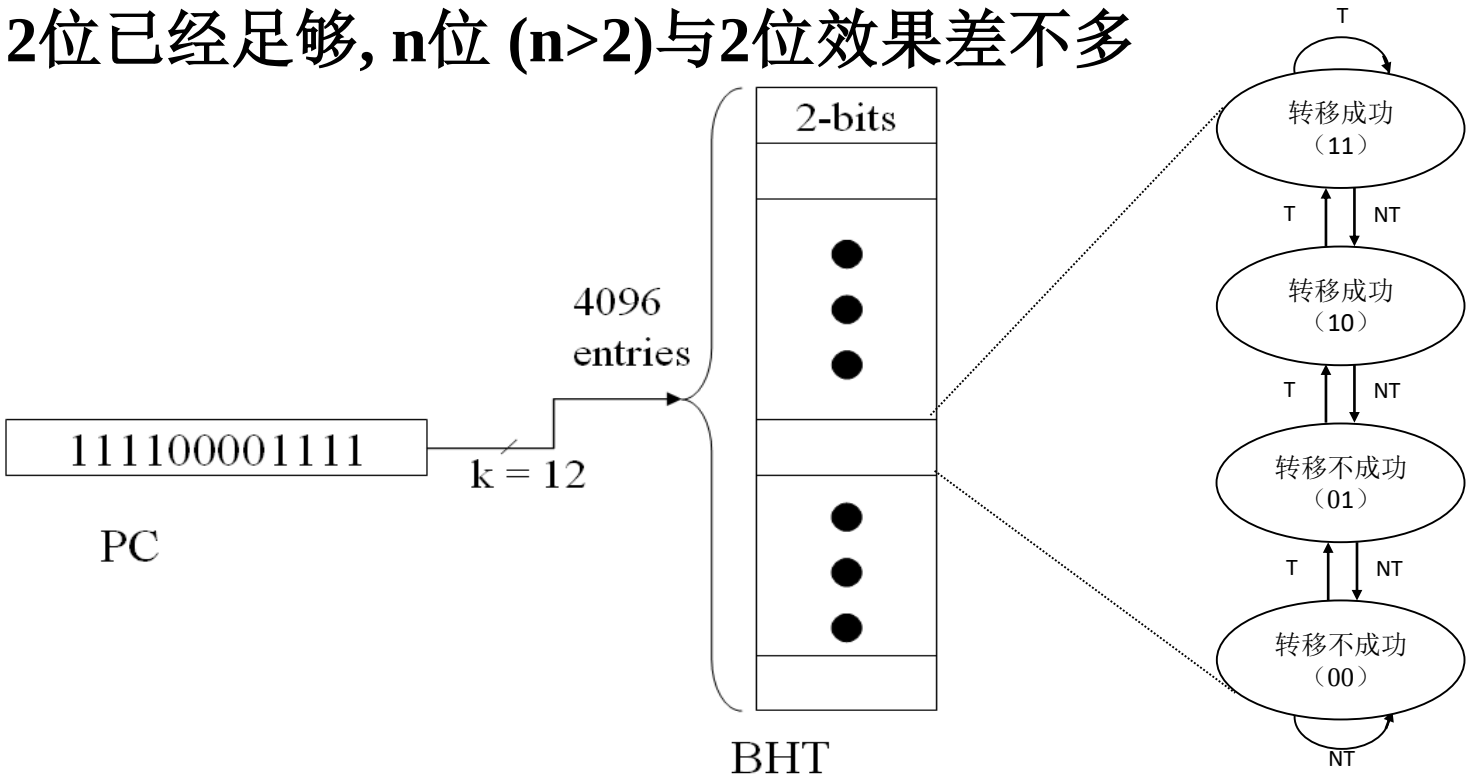
转移预测举例---BHT

- 转移历史表BHT（Branch History Table）
  - 用PC低位索引，不进行地址比较检查（可能有冲突）
  - 每项1位记录同一项上次转移是否成功，表示是否转移成功
  - 例如：for (I=0, I<10; I++){ }，转移模式为$(1111111110)^n$
- 问题
  - 对循环进行猜测时，1位 BHT引起两次猜错
  - 循环退出时，转移方向不一致
  - 进入循环时，和上次退出时的转移方向不一致

for (i=0;i<10;i++) for (j=0; j<10; j++) { …… }

两位BHT表

- 只有连续两次猜错，才会改变猜测方向
  - 在前述两重循环的例子中，内循环预测准确率从80/100 = 80%提高到（7+81）/ 100= 88%
- 4096项已经足够，和无穷项效果差不多
- 2位已经足够, n位 (n>2)与2位效果差不多



转移预测技术举例

- 经典商用处理器转移预测机制

处理器	转移预测机制
Alpha 21064、AMD K5	1位 BHT转移预测
PowerPC604、MIPS R10000	2位 BHT转移预测
Pentium Pro、Pentium II	2级转移预测
Alpha21264	组合转移预测

- 近期商用处理器转移预测机制

处理器	转移预测机制
Intel Ivybridge	BTB (8K-16K?项)；间接目标队列 (? 项)； RAS (? 项)；循环检测
AMD Bulldozer	512 项, 4 路 L1 BTB；5120 项, 5 路 L2 BTB； 512 项 间接目标队列； 24 项 RAS；循环检测
IBM POWER7	8K 项本地 BHT 队列；16K 项全局BHT 队列；8K 项 全局 sel 队列； 128 项间接目标队列； 16 项 RAS
Loongson GS464E	8K项本地 BHT 队列；8K项全局BHT 队列；8K项 全局 sel 队列 128项间接目标队列； 16项 RAS；循环检测

存储器

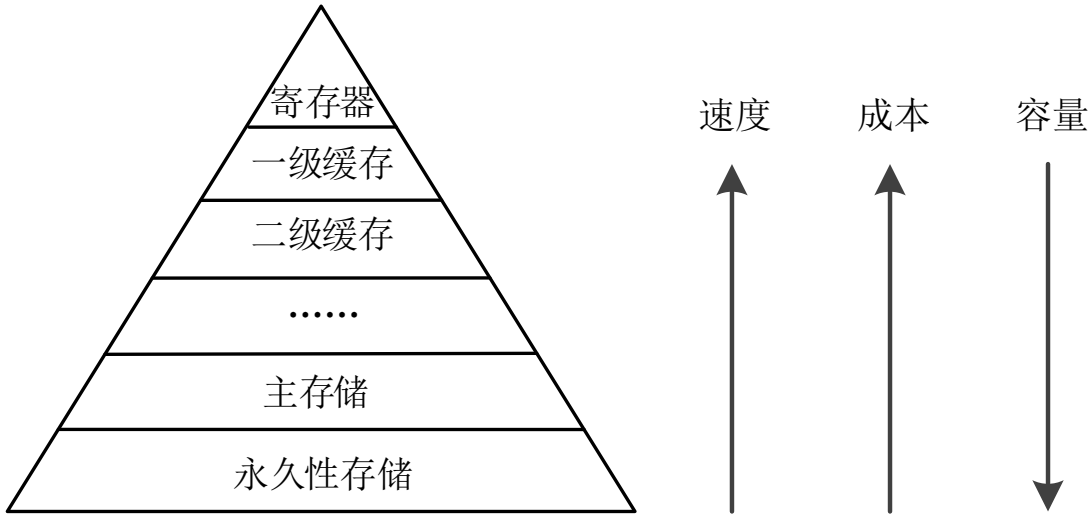
- 存储器存储程序和数据，又称主存储器或内存，一般用动态随机存储器DRAM实现。CPU可以直接访问它，IO设备也频繁地和它交换数据。
  - 存储器的存取速度往往满足不了CPU的快速要求，容量也满足不了应用的需要，为此将存储系统分为高速缓存（Cache)、主存储器和辅助存储器三个层次。
  - Cache存放当前CPU最频繁访问的部分主存储器内容，采用比DRAM速度快但容量小的SRAM实现。数据和指令在Cache和主存储器之间的调动由硬件自动完成。
  - 为扩大存储器容量，使用磁盘、磁带、光盘等大容量存储器作为辅助存储器。计算机运行时所需的应用程序、系统软件和数据等都先存放在辅助存储器中，在运行过程中分批调入主存储器。数据和指令在主存储器和辅助存储器之间的调动由操作系统完成
  - CPU访问存储器时，面对的是一个高速（接近于Cache的速度）、大容量（接近于辅助存储器的容量）的存储器。
  - 现代计算机中还有少量只读存储器（Read Only Memory，简称ROM）用来存放引导程序和基本输入输出系统（Basic Input Output System，简称BIOS）等。
  - 计算机访问内存时采用虚拟地址，操作系统负责维护虚地址和物理地址转换的页表，集成在CPU中的存储管理部件MMU负责把虚拟地址转换为物理地址。

存储——存储介质

- **SRAM （易失、快、贵）**
  - **Cache（高速缓存）**
- **DRAM （易失、慢、便宜）**
  - **DDR SDRAM、Rambus DRAM**
- 闪存（非易失、快、贵）
  - **U盘、SSD**
- 磁性存储介质（非易失、慢、便宜）
  - 硬盘、磁带

存储——存储层次

- 局部性原理
  - 时间局部性
  - 空间局部性
- 访问速度（延迟）
  - 寄存器：一拍多个
  - L1：1-4拍
  - L2：10-20拍
  - L3：40-60拍
  - 内存：100-200拍



高速缓存（Cache）

- 处理器的运算速度和内存容量按摩尔定律的预测指数增加，但内存速度提高非常缓慢，与处理器速度的提高形成了“剪刀差”。访存延迟成为以存储器为中心的冯诺依曼结构的主要瓶颈。
  - **Cache技术利用程序访问内存的时间局部性（一个单元如果当前被访问，则近期很有可能被访问）和空间局部性（一个单元被访问后，与之相邻的单元也很有可能被访问），使用速度较快、容量较小的Cache临时保存处理器常用的数据，使得处理器的多数访存操作可以在Cache上快速进行，只有少量访问Cache不命中的访存操作才访问内存。**
  - **Cache是内存的映像，其内容是内存内容的子集，处理器访问Cache和访问内存使用相同的地址。**
  - **从1980年代开始，RISC处理器就开始在处理器芯片内集成KB级的小容量Cache。现代处理器则普遍在片内集成多级Cache，典型的多核处理器每个处理器核一级指令和数据Cache各几十KB，二级Cache为几百KB，而多核共享的三级Cache为几MB到几十MB。**

Cache性能分析

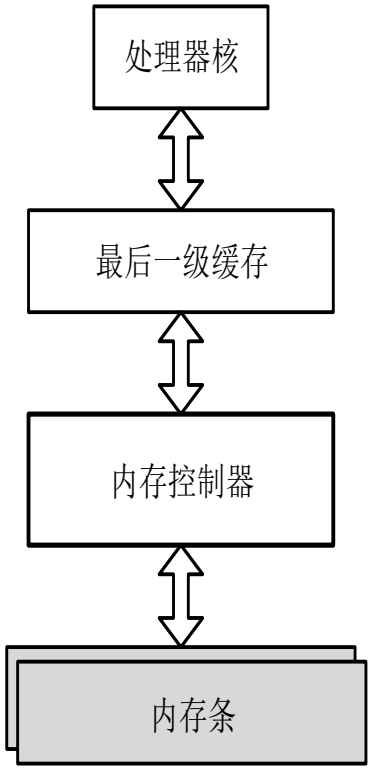
- CPU执行时间与访存延迟的关系
  - AMAT = Average Memory Access Time
  - 如HitTime=1,MissRate=5%,MissPenalty=100，则AMAT = 1+5 =6
- 访存性能优化
  - 降低失效率（MissRate）
  - 降低失效延迟（MissPenalty）
  - 降低命中延迟（HitTime）
  - 提高Cache访问并行性

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

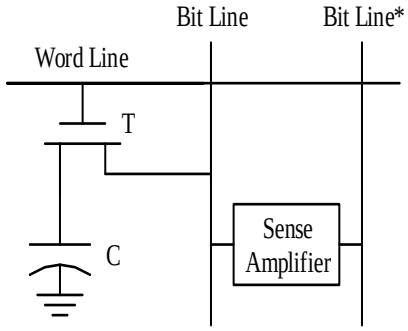
$$AMAT = HitTime + MissRate \times MissPenalty$$

内存

- 冯诺依曼结构中的“存储器”
  - 采用动态随机存储器DRAM，每个单元只要一个晶体管，通过电容充放电保存数据，需动态刷新
  - SDRAM、DDR SDRAM、DDR2 SDRAM、DDR3 SDRAM、DDR4 SDRAM……
- 是CPU性能的决定性因素
  - 容量沿摩尔定律不断变大，但延迟降低非常缓慢（每年7%）
  - 早期内存通过北桥连接到CPU，现代CPU都直接带内存控制器
  - CPU内设置多级Cache来降低平均延迟

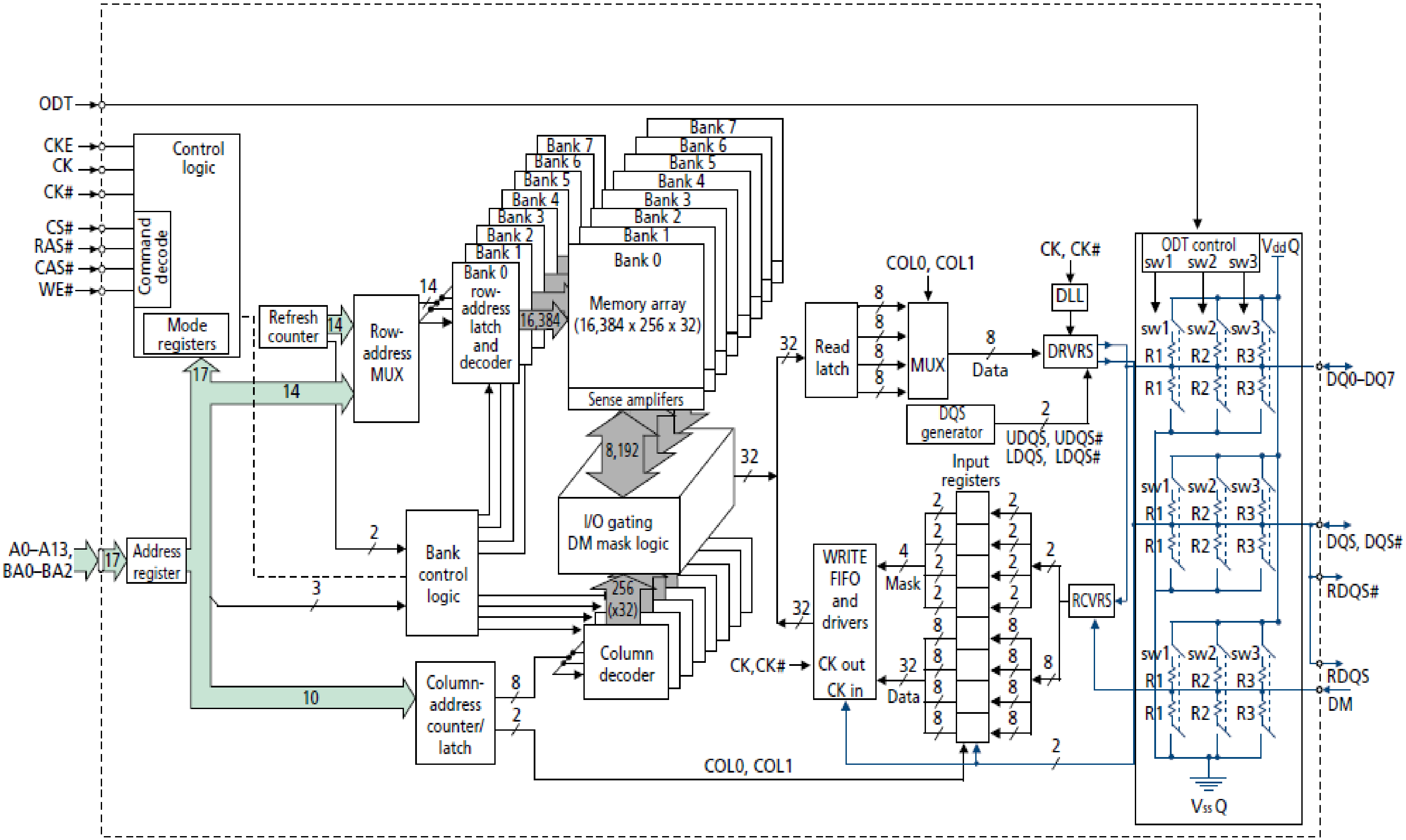


内存基本单元

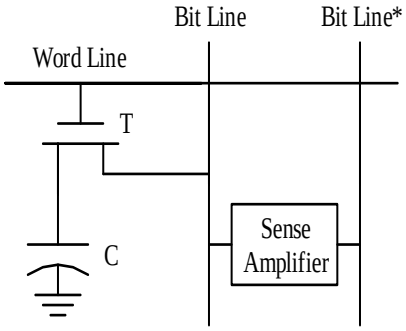


- 由MOS管T和电容（存储单元）组成
  - 电容C存储的电位决定存储单元的逻辑值
  - 字线：根据地址译码，连接同一字的不同位
  - 位线：读写的数据，连接不同字的同一位
- 读写过程
  - 读操作：把位线预充到Vref=VCC/2，字线打开T管，C引起位线微小的电位差，感应放大器读出，此时C中的电位被破坏，感应放大器需要恢复原来的值
  - 写操作：字线上提供更大电流重置感应放大器和位线的值
  - C中的电容可能会漏掉，因此DRAM需要周期刷新，刷新可以通过读操作进行，一般每行几十微秒刷新一次

内存内部结构


ODT
CKE
CK
CK#
CS#
RAS#
CAS#
WE#
Control logic
Command decode
Mode registers
Refresh counter
Row-address MUX
Bank 0 row-address latch and decoder
Bank 1
Bank 2
Bank 3
Bank 4
Bank 5
Bank 6
Bank 7
Bank 0
Memory array (16,384 x 256 x 32)
Sense amplifers
16,384
8,192
I/O gating DM mask logic
256 (x32)
Column decoder
Bank control logic
Column-address counter/ latch
A0–A13, BA0–BA2
Address register
17
14
10
COL0, COL1
Read latch
MUX
Data
DLL
CK, CK#
DRVRS
DQS generator
UDQS, UDQS#
LDQS, LDQS#
Input registers
Mask
WRITE FIFO and drivers
CK out
CK in
RCVRS
ODT control
sw1 sw2 sw3
VddQ
R1 R2 R3
VssQ
DQ0–DQ7
DQS, DQS#
RDQS#
RDQS
DM

内存结构



- 多个Bank（4/8）
  - 每个Bank包含存储阵列和感应放大器
  - 所有Bank共用锁存和写FIFO
- DRAM的读写过程
  - 内存访问并不直接读写存储阵列，而是通过行缓冲进行
  - 数据读到行缓存后，原阵列中就破坏了，需要从行缓存写回（关行）
  - 先将特定行取入行缓冲，再读写行缓冲，最后刷回存储阵列
- 由此产生三种不同行状态：
  - **Close page**：每次读写完都预充，行缓冲不命中，打开 -> 读写
  - **Open page**：行缓冲命中，直接读写
  - **Open page**：行缓冲不命中，关行 -> 打开 -> 读写

提高访存性能的方法

- 利用行缓冲局部性
  - 利用输出端的行缓存，如果两次访问在同一行，可降低访问延迟
  - **Close page：**打开 -> 读写
  - **Open page：**读写（命中）、关行 -> 打开 -> 读写（不命中）
- 利用**Bank**级并行性
  - **DRAM**芯片的多个**Bank**可以流水并行执行
  - 要求连续的访问针对不同**Bank**
- 内存控制器的调度
  - 内存控制器同时对几十个不同的访问进行调度
  - 调度访问次序，利用行缓存局部性和**Bank**级并行性
  - 支持多个**Outstanding**访问操作，读优先……

龙芯3A2000/3A3000访存性能

- 四核双内存通道CPU
  - 主频、相同主频下性能、访存带宽
  - 自主设计的内存控制器比商业内存控制器大幅度提高性能

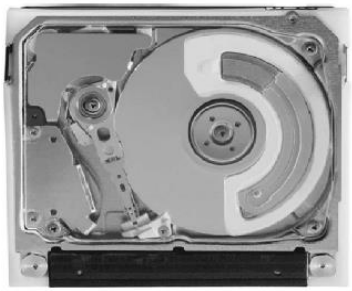
	单核性能			四核性能		
	SPEC INT2006	SPEC FP2006	STREAM（GB）	SPEC INT2006	SPEC FP2006	STREAM（GB）
LS 3A1000（四核,1.0GHz）	2.7	2.5	0.30	9.0	7.7	0.71
LS 3A2000（四核,1.0GHz）	6.9	6.3	6.1	22.5	22.2	9.7
LS 3A3000（四核,1.5GHz）	11.1	10.1	8.8	36.2	32.9	13.2
VIA C-4600（四核，2.0GHz）	10.8	9.8	4.5	27.5	23.2	3.3
AMD K10（四核，1.5GHz）	11.3	11.3	4.5	36.6	34.0	6.0

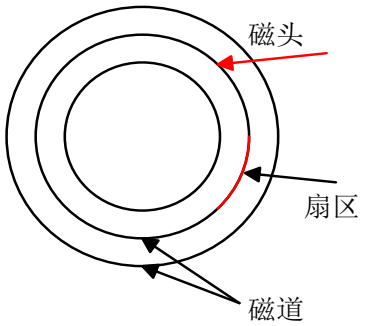
输入/输出设备

- IO设备实现计算机和外部世界的信息交换。传统的IO设备有键盘、鼠标、打印机和显示器等；新型的IO设备能进行语音、图像、影视的输入输出和手写体文字输入，并支持计算机之间通过网络进行通信；磁盘等辅助存储器在计算机中也当作IO设备来管理。处理器通过读写IO设备控制器中的寄存器来访问及控制IO设备。高速IO设备可以在处理器安排下直接与主存储器成批交换数据，称为直接存储器访问（Directly Memory Access，简称DMA）。处理器可以通过查询设备控制器状态与IO设备进行同步，也可以通过中断与IO设备进行同步。

硬盘

- 永久性大容量存储设备
  - 构造原理为：将磁性材料覆盖在圆形碟片上，通过一个读写头（磁头）悬浮在碟片表面来感知存储的数据。
  - 容量大，价格低，顺序访问
  - 访问时间ms级：先磁道，再扇区，与转速有关（7200转、5400转）
  - 通过磁盘阵列提高吞吐率，如RAID
- 与CPU可以通过PIO也可以通过DMA方式传输数据
  - 磁盘控制器中有缓存，在缓存命中速度高很多





闪存（Flash）

- 非易失性半导体存储器
  - 可以随机访问
  - 访问延迟只有磁盘的千分之一到百分之一
  - 但容量比磁盘小，每GB价格高
- NOR Flash和NAND Flash
  - NOR Flash容量小，写入慢，但可靠性高，一般用作BIOS存储
  - NAND Flash容量大，存储单位随擦写次数多容易损坏，可通过地址重映射方式来分布写操作，称为磨损均衡（Wear Leveling），平均擦写可达10万次，U盘、SD卡、SSD固态硬盘均采用NAND Flash
  - NAND Flash比NOR Flash更容易出位交换错，需要ECC算法纠正

GPU（Graphic Processing Unit）

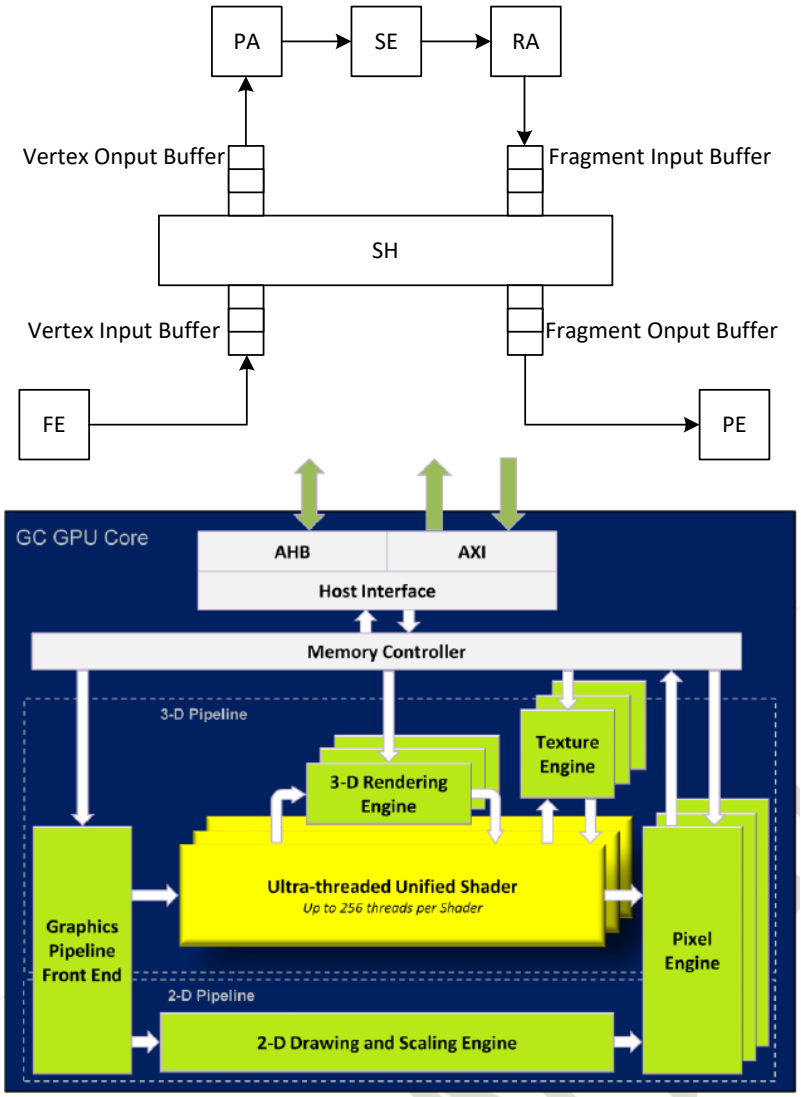
- **GPU是和CPU之间联系最紧密的“外设”**
  - **NVIDIA、Imagination、ATI（AMD）、Intel、ARM**
  - **世界上凡是不做GPU的CPU企业都活不好**
- **CPU（内存）和GPU（显存）之间的数据传输**
  - **CPU把要显示的原始数据放在内存，有些数据要通过PIO写到显存**
  - **GPU通过DMA把原始数据从内存搬到显存**
  - **CPU通过Uncache Accelerate技术把多个连续的写操作以加速写显存**
  - **显存中的帧缓存（Frame Buffer）需要定时访问以刷新屏幕（如每秒60帧），并具有最高的访问优先级**
  - **APU把显存和内存合并，可以减少CPU和GPU的数据搬运，但增加了内存的负担（GPU在计算时也需要访问内存）**

GPU的内部结构

- 功能：图形渲染流水线的硬件加速实现
  - 顶点读入VF，从内存/显存中取出顶点信息（位置/颜色等）
  - 顶点渲染VS，对每一个顶点，进行坐标变换、颜色计算
  - 图元装配PA，将顶点组合成图元（三角形等）
  - 光栅化RS，矢量图形转换为点阵
  - 像素渲染FS，计算每个像素的颜色（纹理采样）
  - 帧缓冲操作FOP，进行深度测试，计算最终的像素颜色
- 实现：
  - 专用硬件：VF、PA、RS、FOP
  - 可编程众核：VS、FS

GPU北部结构

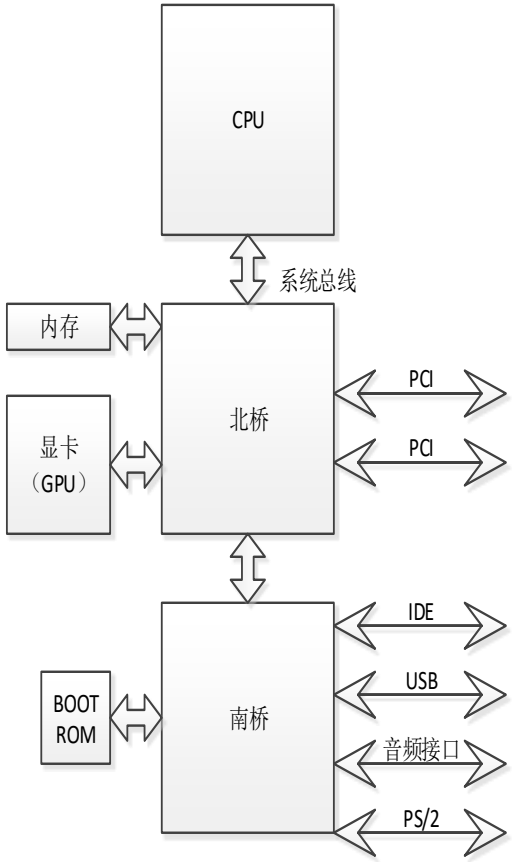
- **Front End**
  - 取命令、顶点数据
- **Unified Shader**
  - 运行VS/FS代码
- **3D Rendering Engine**
  - 图元装配、光栅化
- **Texture Engine**
  - 纹理采样
- **Pixel Engine**
  - 像素处理



计算机硬件结构的演进

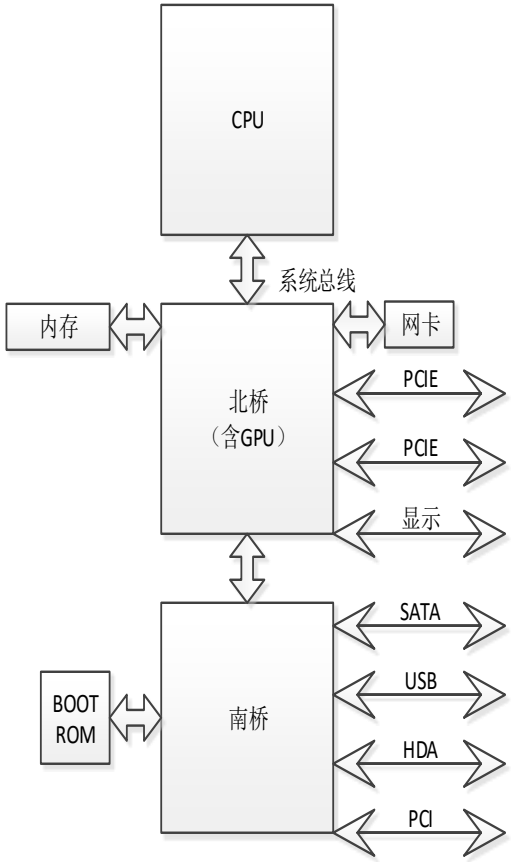
CPU+GPU+北桥+南桥

- 处理器+芯片组时代
  - 在此之前是分离元件时代
- 计算机硬件结构四大组件
  - **CPU、北桥、GPU、南桥**
  - 内存连在北桥上
  - 北桥是系统连接的枢纽
  - 南桥连接低速IO设备



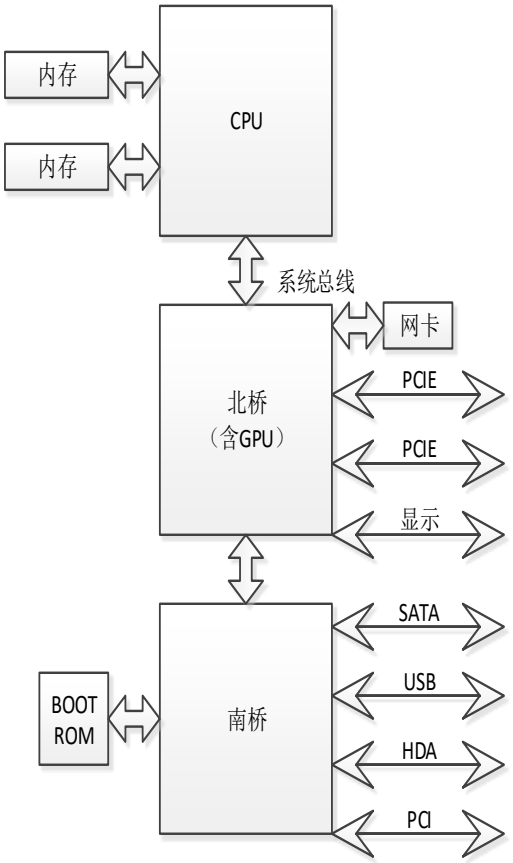
CPU+北桥+南桥

- **GPU被集成到北桥中**
  - **用于对图形性能要求不是特别高的场景，如桌面办公、上网本等**



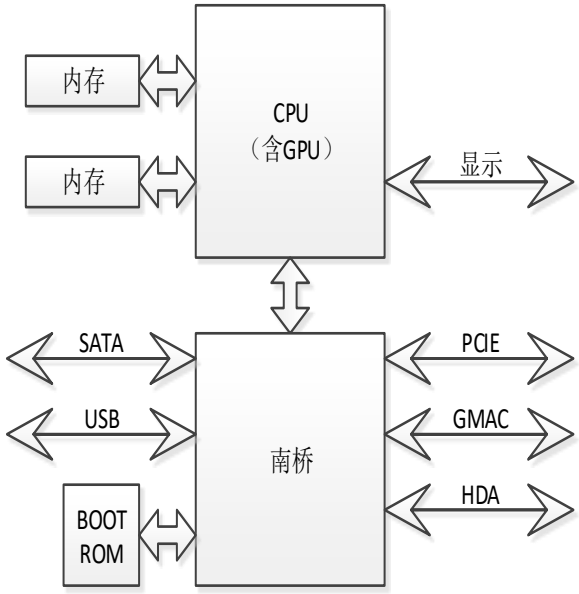
CPU+弱北桥+南桥

- 内存控制器被集成到CPU中
  - 大大缓解了冯诺依曼结构带来的缓存瓶颈，平均可带来30%的性能提高
  - 降低了对系统总线的需求（多路服务器对系统总线要求除外），使Intel系统总线专利减少了用武之地



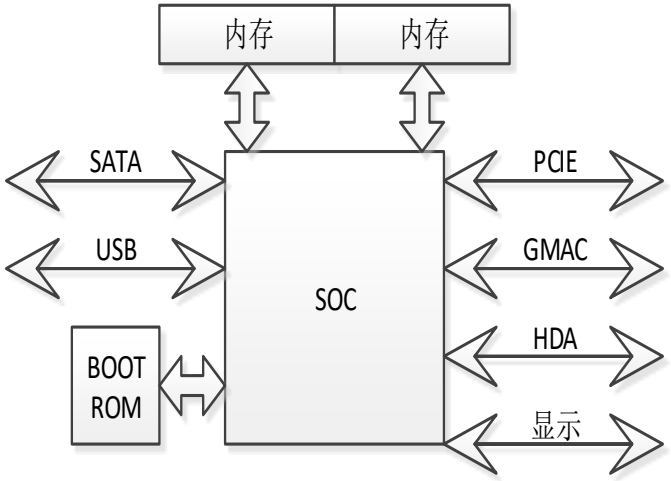
CPU+南桥

- GPU被集成到处理器中
  - 北桥的功能进一步减弱，和南桥合并，形成处理器+南桥两片结构
  - GPU集成到处理器中减少了从内存到显存的数据传输，但显存合并到内存增加了内存压力，GPU对内存带宽要求很高，且实时性要求高
- 内存控制器集成在CPU中，GPU集成在北桥中的两片方案也是一个不错的选择
  - 独立显存有利于性能
  - CPU升级方便



SOC单片方案

- 所有的接口都集成到一个芯片上，形成片上系统SoC（System on Chip）
  - 单片系统降低了系统设计灵活性，一般用于中低端设备和移动设备
- 主要CPU厂商中，高端CPU采用两片方案，低端CPU采用单片方案
  - Intel的酷睿/至强系列
  - Intel的凌动系列



处理器和IO间通信

IO寄存器寻址

- **IO访问与存储访问的不同**
  - **存储器是存储单元阵列，存储访问通过读写指令直接完成，对某单元的读写不会影响其它单元**
  - **IO设备都有专门的设备控制器，设备控制器向CPU提供一组IO寄存器，CPU通过读取IO寄存器获知IO控制器状态，通过写（有时候是读）IO寄存器来控制IO设备，CPU写入IO寄存器的数据，会被设备控制器解释成控制IO设备的命令**
- **IO寄存器寻址**
  - **通过独立的IO指令寻址，如X86的IN和OUT指令**
  - **内存映射统一寻址，如MIPS通过LW和SW指令的地址区分是内存访问还是IO访问**

CPU和IO设备间的同步

- **CPU和IO控制器是两个不同的主体，两者协同完成IO访问，**
- 查询方式
  - **CPU通过不断读取IO状态寄存器的内容获取设备控制器的状态**
  - **如打印机控制器包括状态寄存器和控制寄存器，CPU在打印一串数据时，先把数据写入数据寄存器，然后不断读取状态寄存器的值，当读出的“完成位”为“1”时，再把下一个数据写入数据寄存器**
- 中断方式
  - **查询方式效率太低，改由设备完成某个操作时，产生中断通知CPU，把CPU从查询IO中解放出来，提高CPU的利用率**
  - **如CPU写入打印机的数据寄存器后，转去执行别的操作，打印机打印完数据寄存器的数据后，通过中断通知CPU，CPU再查询状态寄存器**

存储器和IO设备间的通信

- 存储器和IO设备之间需要大量的通信
  - 系统启动时，需要操作系统代码和数据从硬盘搬运到内存
  - 显示输出时，需要把显示数据从内存搬运到GPU的显存
  - 冯诺依曼结构本质上是以内存为中心的结构
- 存储器和IO设备间通信有两种方式
  - PIO（ Programming Input/Output）方式：CPU从IO设备/内存中把数据读到CPU内部寄存器中，再写入到内存/IO设备
  - DMA（Direct Memory Access）方式：在内存和外设之间开辟直接的数据传输通道，由DMA控制器控制数据在内存和外设之间直接、连续传输

PIO传输方式

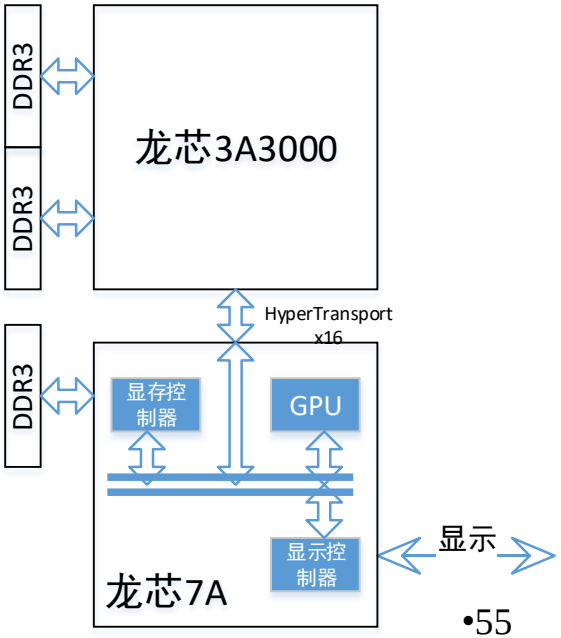
- **IO设备和内存之间的通信通过CPU的寄存器中转**
  - **CPU和IO设备之间的同步可以是查询方式，也可以是中断方式**
- **使用PIO传输方式的设备**
  - **计算机中多数设备都可以通过PIO访问**
  - **低速设备如键盘、鼠标只能通过PIO访问**
- **CPU访问IO设备都是Uncache访问**
  - **提供Uncache Accelerate加速：把多个连续访问合并成一个大的**
  - **对很多IO设备的性能至关重要，如CPU往GPU传数据时，部分数据通过PIO方式传输**

DMA数据传输

- 在存储器和外设之间开辟直接的数据传送通道，数据传送由专门的硬件（DMA控制器）来控制。
  - 处理器准备某内存区域为DMA区域
  - 处理器设置DMA控制器参数后可以去干别的事
  - DMA控制器进行数据传输
  - DMA控制器向处理器发出一个中断，通知处理器数据传送的结果
  - 处理器收到中断后安排下一块传输
- DMA要Cache和内存的一致性
  - DMA前CPU先刷Cache，现代处理器一般由硬件自动维护一致性
- 多数高速IO设备均采用DMA传输方式
  - 硬盘、网络

CPU、GPU与DC间的数据传输

- 龙芯3A3000+龙芯7A桥片两片结构
  - GPU与DC（显示控制器）在桥片中集成
  - 专供GPU、DC使用的显存控制器在桥片中集成
- CPU、GPU与DC之间的几种数据传输方式
  - PIO方式
    - CPU读写GPU中的控制寄存器
    - CPU读写DC中的控制寄存器
    - CPU读写显存
  - DMA方式
    - GPU读写内存/显存
    - DC读内存/显存

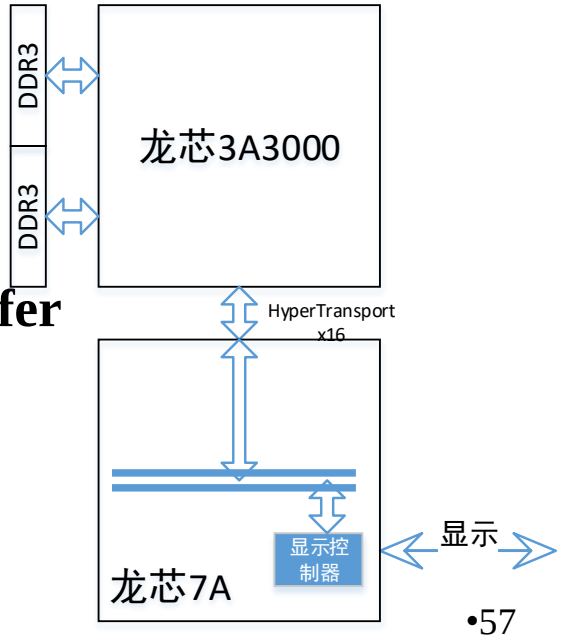


GPU与DC间的DMA差异

- **DC的DMA行为比较简单**
  - **DC的作用是将画面进行持续显示，无需进行复杂的计算**
  - **DC的DMA初始化主要是将显示分辨率，帧缓冲（framebuffer）指针配置好。然后开始周期性地DMA数据搬运。**
  - **CPU或GPU在这一过程中会周期性地向一个帧缓冲或多个帧缓冲（与DC的实现相关）进行填充，填充时机通常是由DC的中断决定**
- **GPU的DMA行为稍微复杂**
  - **只有需要的时候，CPU才会调用GPU进行运算**
  - **GPU的初始化主要是描述符列表的指针，描述符是在内存里规定好的数据结构。每个描述符中又包含指向数据区域的指针**
  - **CPU在需要的时候，会将数据区域填好，修改相应的描述符，再通知GPU启动DMA。GPU通过描述符得到数据区域，进行相应计算。**

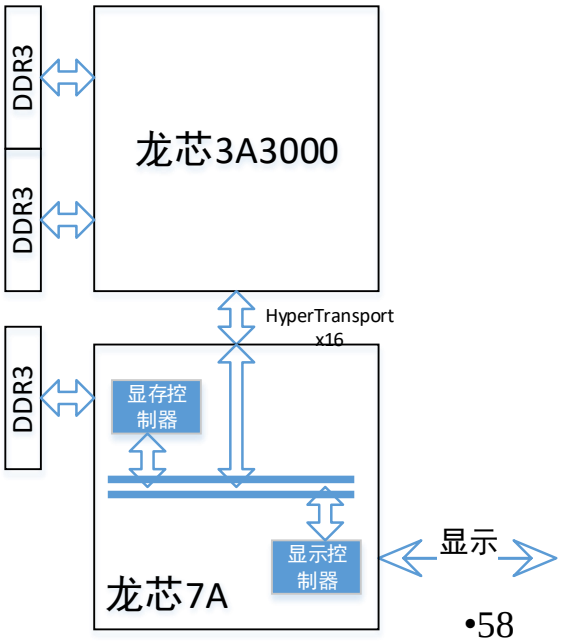
模式一：CPU与DC使用共享显存

- 不使用GPU
- 共享显存
  - 不使用桥片上的显存，而在内存中分配一个区域专供显示使用
  - 这个区域称之为framebuffer（帧缓存）
- 数据传输方式
  - CPU读写DC中的控制寄存器，启动DMA
    - PIO操作
  - CPU将需要显示的内容写入内存framebuffer
    - CPU正常内存操作
  - DC读内存framebuffer
    - DMA操作



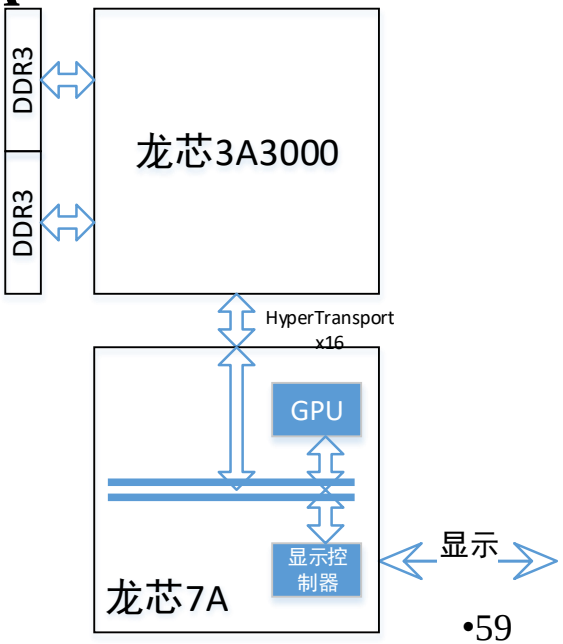
模式二：CPU与DC使用独立显存

- 不使用GPU
- 独立显存
  - 使用桥片上的显存
  - 这个区域称之为framebuffer（帧缓存）
- 数据传输方式
  - CPU读写DC中的控制寄存器，启动DMA
    - PIO操作
  - CPU将需要显示的内容从内存读出，再写入独立显存上的framebuffer
    - PIO操作
  - DC读显存framebuffer
    - 桥片内的访存显存操作



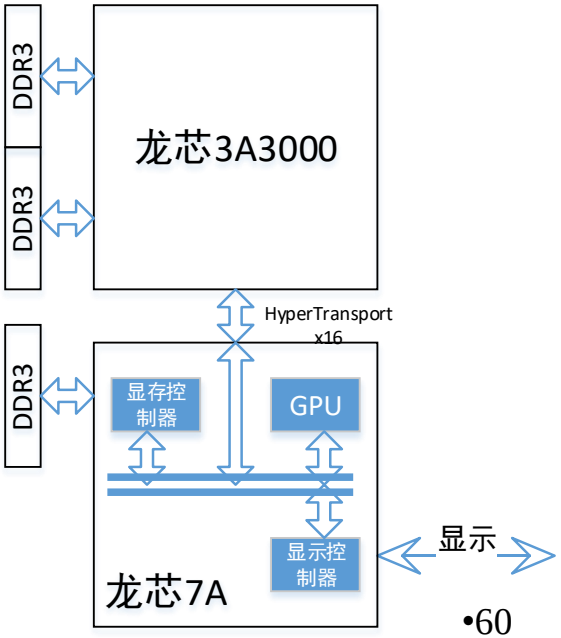
模式三：CPU、GPU/DC使用共享显存

- 数据传输方式
  - **CPU读写DC中的控制寄存器，启动DMA**
    - **PIO操作**
  - **CPU在内存中分配GPU使用的空间，并将相关数据填入**
  - **CPU读写GPU中的控制寄存器，启动DMA**
    - **PIO操作**
  - **GPU读内存**
    - **DMA操作**
  - **GPU将计算结果写入内存framebuffer**
    - **DMA操作**
  - **DC读内存framebuffer**
    - **DMA操作**



模式四：CPU、GPU/DC使用独立显存

- 数据传输方式
  - **CPU读写DC中的控制寄存器，启动DMA**
    - **PIO操作**
  - **CPU在内存中分配GPU使用的空间，并将相关数据填入**
  - **CPU读写GPU中的控制寄存器，启动DMA**
    - **PIO操作**
  - **GPU读内存**
    - **DMA操作**
  - **GPU将计算结果写入显存framebuffer**
    - **桥片内的访存显存操作**
  - **DC读内存framebuffer**
    - **桥片内的访存显存操作**



几种使用模式比较

- 无GPU的共享显存与独立显存
  - 对CPU来说，写内存的性能比写IO性能高得多
  - 在DC读内存（DMA）带宽能够保证的前提下，共享显存性能更好
- 有GPU的共享显存与独立显存
  - 独立显存的使用能够减少两次DMA操作，有效降低内存带宽需求
- 有GPU与无GPU的比较
  - GPU对于复杂的图形变换计算效率较高，尤其是处理3D图形时
  - CPU在处理某些简单图形时会更快，并且节省了DMA启动开销
- 实际使用中，即使有GPU，也会存在一些显示数据由CPU直接写入framebuffer的情况，所以PIO性能也会对系统性能造成一定影响

IO中断控制器

- 中断：打断当前CPU的执行进程，转去执行某特定程序
  - 中断源发出中断信号到中断控制器
  - 中断控制器产生中断请求给CPU
  - CPU响应中断并读取中断类型码
  - CPU根据中断类型执行相应的中断服务程序
  - CPU从中断服务程序返回
- 中断信号的传递
  - 中断线直传和MSI（Message Signaled Interrupt）
- Intel 8259A的中断寄存器
  - 中断请求寄存器（IRR）：存放当前的中断请求
  - 中断在服务寄存器（ISR）：存放正在服务的中断请求
  - 中断屏蔽寄存器（IMR）：存放中断屏蔽位

PIO与DMA

键盘输入（PIO）	网络收包（DMA）
敲击键盘	接收端看到网络包
键盘输入被记录在PS/2控制器内	网卡将收到的网络包写入内存中预先分配好的区域
中断CPU	中断CPU
CPU查询中断源，发现键盘中断	CPU查询中断源，发现网络中断
CPU从南桥的PS/2控制器读键盘值	CPU从内存中读网络包，初始化新的接收缓冲区供网卡使用
CPU清中断	CPU清中断

常见处理器的结构参数

		Intel Ivybridge	AMD BullDozer	IBM Power7	GS464E
前端	一级指令缓存	32KB，8 路，64B/行	64KB，2路，64B/行	32KB，4 路，128B/行	64KB，4 路，64B/行
	指令TLB	128 项，4 路，L1 ITLB	72 项，全相联，L1 ITLB 512 项，4 路，L2 ITLB	64 项，2 路，L1 ITLB	64 项，全相联，L1 ITLB
	分支预取	BTB（8K-16K?项） 间接目标队列（? 项） RAS（? 项） 循环检测	512 项，4 路 L1 BTB 5120 项，5 路 L2 BTB 512 项 间接目标队列 24 项 RAS 循环检测	8K 项本地 BHT 队列， 16K 项全局BHT 队列， 8K 项 全局 sel 队列 128 项间接目标队列 16 项 RAS	8K项本地 BHT 队列 8K项全局BHT 队列 8K项 全局 sel 队列 1K项间接目标队列 16项 RAS 循环检测
乱序执行	ROB	168 项	128 项	120 项	128 项
	发射队列	54-项 统一	60-项 浮点(共享) 40-项 定点，访存	48-项 标准	32-项 浮点； 32-项 定点； 32-项 访存
	寄存器重命名	160 定点； 144 浮点	96 定点； 160 浮点(双核共享)	80 定点,浮点； 56 CR； 40 XER， 24 Link&Count	128 定点； 128 浮点/向量； 16 Acc； 32 DSPCtrl；32 FCR
运算部件	执行单元	ALU/LEA/Shift/128位 MUL/128位 Shift/256位FMUL/256位 Blend + ALU/LEA/Shift/128位 ALU /128bit Shuffle/256位 FADD + ALU/Shift/Branch/ 128位 ALU/128bit Shuffle/256位 Shuffle/256位 Blend	ALU/IMUL/Branch + ALU/IDIV/Count + 128位FMAC/128位 IMAC + 128位FMAC/128位 XBAR + 128位 MMX + 128位 MMX/128位 FSTO	2 定点 + 2 浮点/向量 + 1 转移 + 1 CR	2 定点/转移/DSP + 2 浮点/向量
	向量宽度	256位	128位	128位	256位

常见处理器的结构参数

		Intel Ivybridge	AMD BullDozer	IBM Power7	GS464E
访存	访存单元个数	2 取+ 1 存	2 取/存	2 取/存	2 取/存
	Load/Store队列	64-项 Load 队列, 36-项 Store队列	40-项 Load 队列, 24-项 Store 队列	32-项 Load队列, 32-项 Store 队列	64-项 取/存 队列
	Load/Store 宽度	128 位	128 位	256 位 load, 128 位 Store	256 位
	TLB	100项全相联, L1 DTLB, 512项4路, L2 TLB	32项全相联, L1 DTLB, 1024项8路, L2 TLB	64项全相联, L1 DTLB, 512项4路, L2 TLB	32-项全相联L1DTLB, 每项两页 1024项8路L2 TLB, 每项两页
	L1D	32KB, 8 路, 64B/行	16KB, 4路, 64B/行	32KB, 8 路, 128B/行	每核64KB, 4 路, 64B/行
	L2	每核256KB, 8路, 64B/行	双核共享2MB, 16 路	每核256KB, 8路, 28B/行	每核256KB, 16路,64B/行
	LLC	8个核20 MB	4个核8 MB	8个核32 MB	4个核4 MB～8MB
	L1失效队列	10	?	8	Unified 16
	L2失效队列	16	23	24	
	L1 Load-to-use	定点4时钟周期, 浮点/向量5时钟周期	4时钟周期	定点2时钟周期, 浮点/向量3 时钟周期	定点3时钟周期, 浮点/向量5 时钟周期
	L2 Load-to-use	12 时钟周期	18-20 时钟周期	8	22 时钟周期
	多层次硬件预取	有	有	有	有

作 业