



中国科学院大学
University of Chinese Academy of Sciences

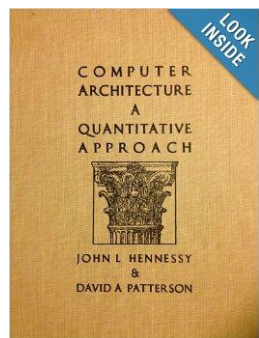


高级计算机系统结构

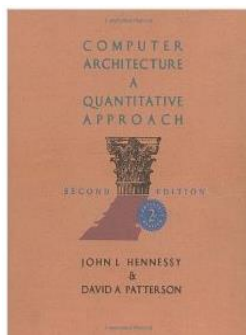
沈海华

shenhh@ucas.ac.cn

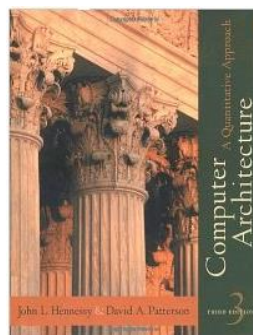
体系结构30年



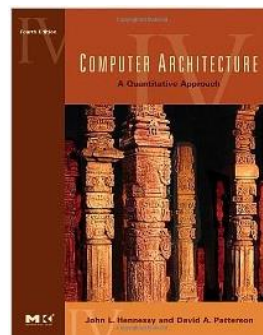
1990



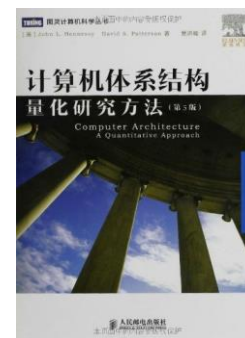
1996



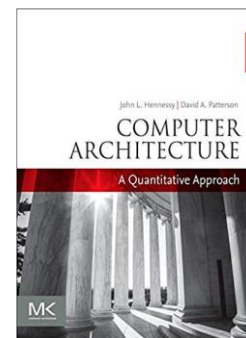
2002



2006



2011

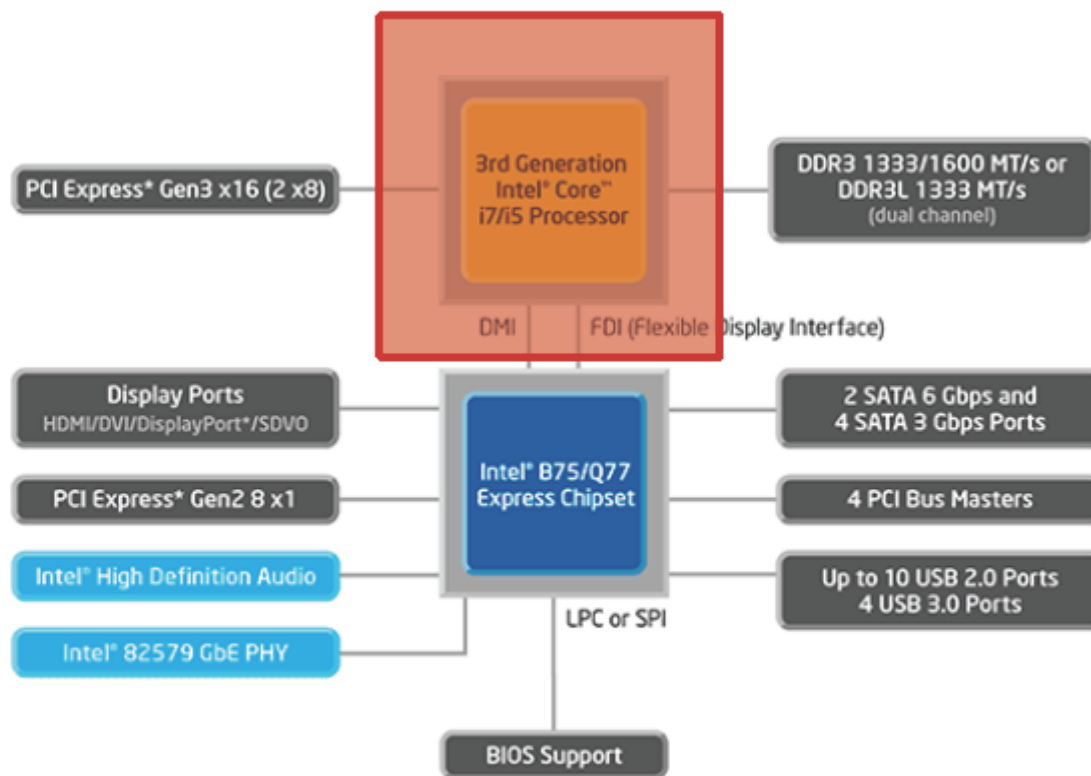


2017

- 强调流水线、**ILP**等并行技术；
- 强调功耗；
- 强调存储系统结构设计；
- 关注向量、**SIMD**和**GPU**等大数据流并行结构；
- 关注面向互联网的数据中心系统结构设计；
- 关注面向移动计算的硬件系统结构设计；
- 关注面向领域的硬件系统结构设计**DSA**

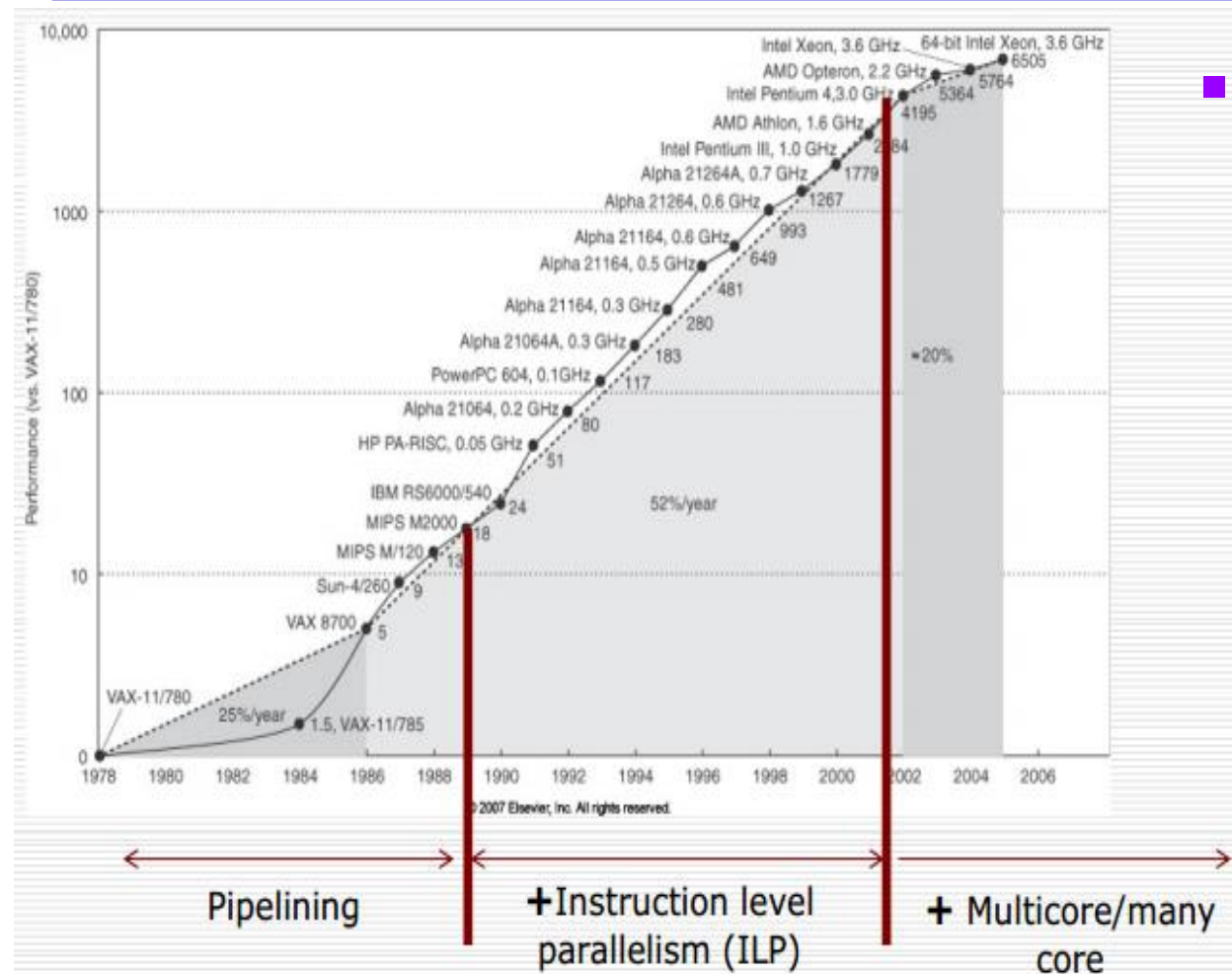
现代计算机系统结构示意图

■ System diagram of a modern system:



■ Today's lecture: focus on processor chip

CPU主要体系结构改进



■ 体系结构经典技术：

- 流水线
- 动态调度
- Cache
- 向量机
- RISC
- 多发射&乱序执行
- EPIC
- SMT
- CMP

第二讲 经典技术回顾：处理器核设计基础


- 处理器核设计技术及其量化分析基础
 - 性能参数计算
 - 现代处理器核设计技术基础
 - Pipelines
 - Branch Prediction
 - Register Renaming
 - Out-of-Order Execution
 - Re-order Buffer
 - Data Parallel Processing:SIMD/Vector Extensions

程序执行时间

- Latency metric: program execution time in seconds

$$\begin{aligned} CPUtime &= \frac{Seconds}{Program} = \frac{Cycles}{Program} \cdot \frac{Seconds}{Cycle} \\ &= \frac{Instructions}{Program} \cdot \frac{Cycles}{Instruction} \cdot \frac{Seconds}{Cycle} \\ &= IC \cdot CPI \cdot CCT \end{aligned}$$

- Your system architecture can affect all of them

- CPI: memory latency, IO latency, ...
 - CCT: cache org., power budget, ...
 - IC: OS overhead, compiler choice ...
- 
- Independent?

程序执行时间的进一步分解

- $\text{Execution Time} = \text{IC} \cdot \text{CPI} \cdot \text{CCT}$
 - $\text{IC} = \text{instruction count}$
 - $\text{CPI} = \text{cycles per instruction} (= 1/\text{IPC})$
 - $\text{CCT} = \text{clock frequency} (= 1/\text{Frequency})$
- $\text{CPI} = \text{CPI}_{\text{base}} + \text{CPI}_{\text{stalls}}$
 - Stalls due to data hazards
 - RAW, WAW, WAR dependencies
 - Stalls due to control hazards
 - Resolving jumps and branches
 - Stalls due to memory latency
 - Large memories are slow

性能的另一面：功耗

- $\text{Power} \cong C \cdot V^2 \cdot F + V_{\text{dd}} \cdot I_{\text{leakage}}$
 - Dynamic power when calculating
 - C capacitance of transistors, V power supply, F frequency
 - Static power when idling
 - V power supply, I_{leakage} the leakage current of transistors
- Power/area
- $\text{Energy} = \text{Power} \cdot \text{Execution Time}$
- Energy/instruction

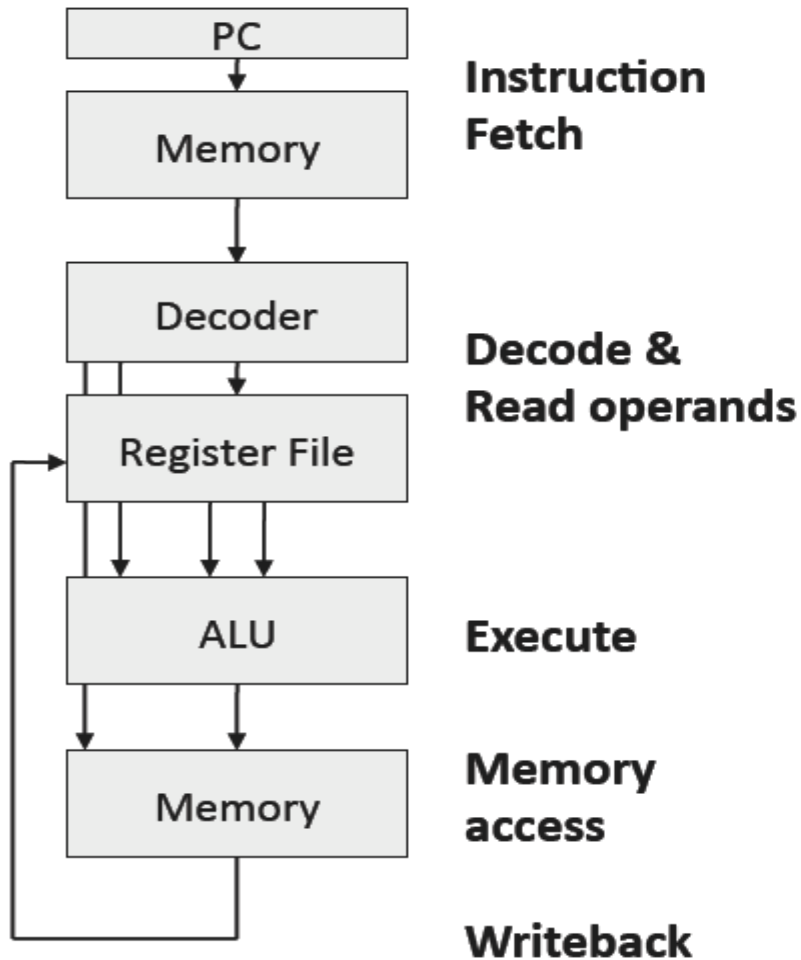
第二讲 走进处理器核设计：经典技术回顾

- 处理器核设计技术及其量化分析基础
 - 性能参数计算
 - 现代处理器核设计技术基础
 - Pipelines
 - Branch Prediction
 - Register Renaming
 - Out-of-Order Execution
 - Re-order Buffer
 - Data Parallel Processing:SIMD/Vector Extensions

计算机系统中常见的指令类型

- Arithmetic & logical
 - Input and outputs are registers
- Loads and stores
 - Move data between registers and memory
- Control flow
 - Change order of execution within a program
 - Unconditional (jumps) or conditional (branches)

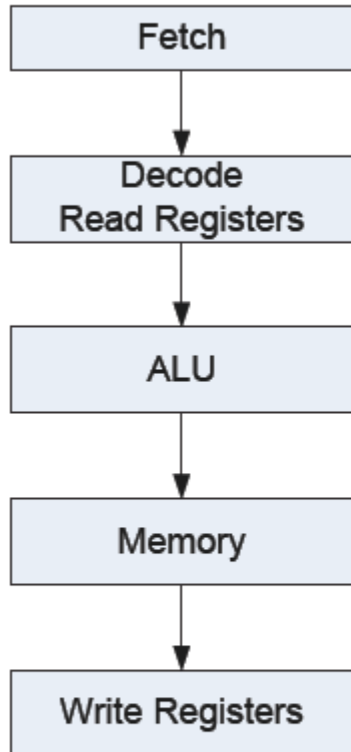
Single-Cycle Processor



■ Single-cycle design

- Direct interpretation of ISA
- One cycle per instruction
- One instruction at the time

5-stage Pipelined Processors(MIPS R3000, 1985)



■ Advantages

- CPI_{base} is 1 (pipelining)
- No WAW or WAR hazards
- Simple and elegant
 - Still used in many ARM & MIPS designs

■ Shortcomings

- Upper performance bound is $CPI=1$
- High-latency instructions not handled well
 - 1 stage for accesses to large caches or multiplier
 - Clock cycle is high
- Unnecessary stalls due to rigid pipeline
 - If one instruction stalls anything behind it stalls

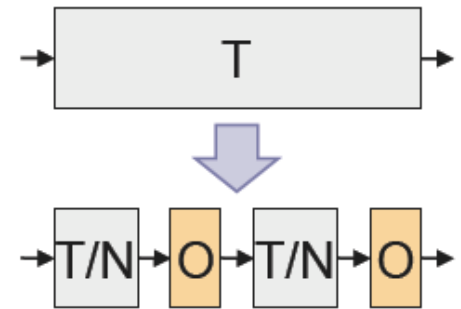
流水线技术的改进

- Higher clock frequency (lower CCT): deeper pipelines
 - Overlap more instructions
- Lower CPI_{base} : wider pipelines
 - Insert multiple instruction in parallel in the pipeline
- Lower CPI_{stall} :
 - Diversified pipelines for different functional units
 - Out-of-order execution
- Balance conflicting goals
 - Deeper & wider pipelines \Rightarrow more control hazards
 - Branch prediction (speculation)
- It all works because of instruction-level parallelism (ILP)

流水线深度限制

■ Each pipeline stage introduces some overhead (O)

- Delay of pipeline registers
- Inequalities in work per stage
 - Cannot break up work into stages at arbitrary points
- Clock skew
 - Clocks to different registers may not be perfectly aligned



■ If original CCT was T , with N stages CCT is $T/N+O$

- If $N \rightarrow \infty$, $\text{speedup} = T / (T/N+O) \rightarrow T/O$
 - Assuming that IC and CPI stay constant
- Eventually overhead dominates and leads to diminishing returns

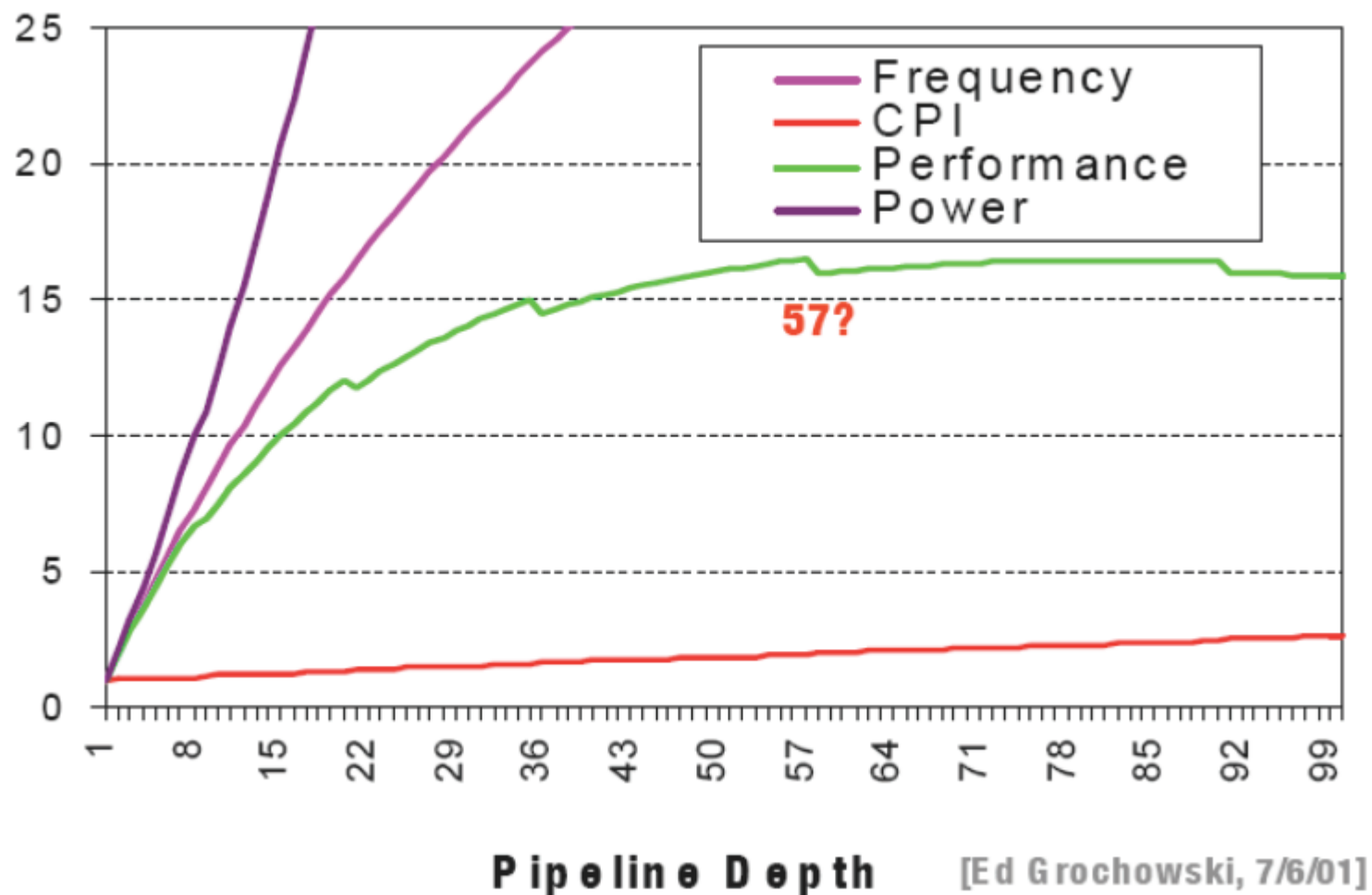
加速比与Amdahl定律

- $\text{Speedup} = \text{CPUtime}_{\text{old}} / \text{CPUtime}_{\text{new}}$
- Given an optimization x that accelerates fraction f_x of program by a factor of S_x , how much is the overall speedup?

$$\text{Speedup} = \frac{\text{CPUTime}_{\text{old}}}{\text{CPUTime}_{\text{new}}} = \frac{\text{CPUTime}_{\text{old}}}{\text{CPUTime}_{\text{old}}[(1 - f_x) + \frac{f_x}{S_x}]} = \frac{1}{(1 - f_x) + \frac{f_x}{S_x}}$$

- Lesson's from Amdhal's law
 - Make common cases fast: as $f_x \rightarrow 1$, $\text{speedup} \rightarrow S_x$
 - But don't overoptimize common case: as $S_x \rightarrow \infty$, $\text{speedup} \rightarrow 1 / (1 - f_x)$
 - Speedup is limited by the fraction of the code that can be accelerated
 - Uncommon case will eventually become the common one
 - Amdhal's law applies on other metrics too: cost, power consumption, ...

流水线与功耗限制



在过去很长一段时间，切分更深的流水线、提高主频是提高处理器性能的主要手段

■ Advantages: Higher clock frequency

- The workhorse behind multi-GHz processors
- Opteron: 11; UltraSparc: 14; Power5: 17; Pentium4: 22/34; Nehalem: 16; ...

■ Cost

- Complexity: more forwarding & stall cases

■ Disadvantages

- More overlapping → more dependencies → more stalls
 - CPIstall grows due to data and control hazards
- Clock overhead becomes increasingly important
- Power consumption

超标量流水线（多发射）

■ Advantages: lower CPI_{base} ($1/N$)

- Opteron: 3, UltraSparc: 4, Power5: 8, Pentium4: 3; Core 2: 4; Nehalem: 4; ...

■ Cost

- Need wider path to instruction cache
- Need more ALUs, more register file ports, ...
- Complexity: more forwarding & stall cases to check

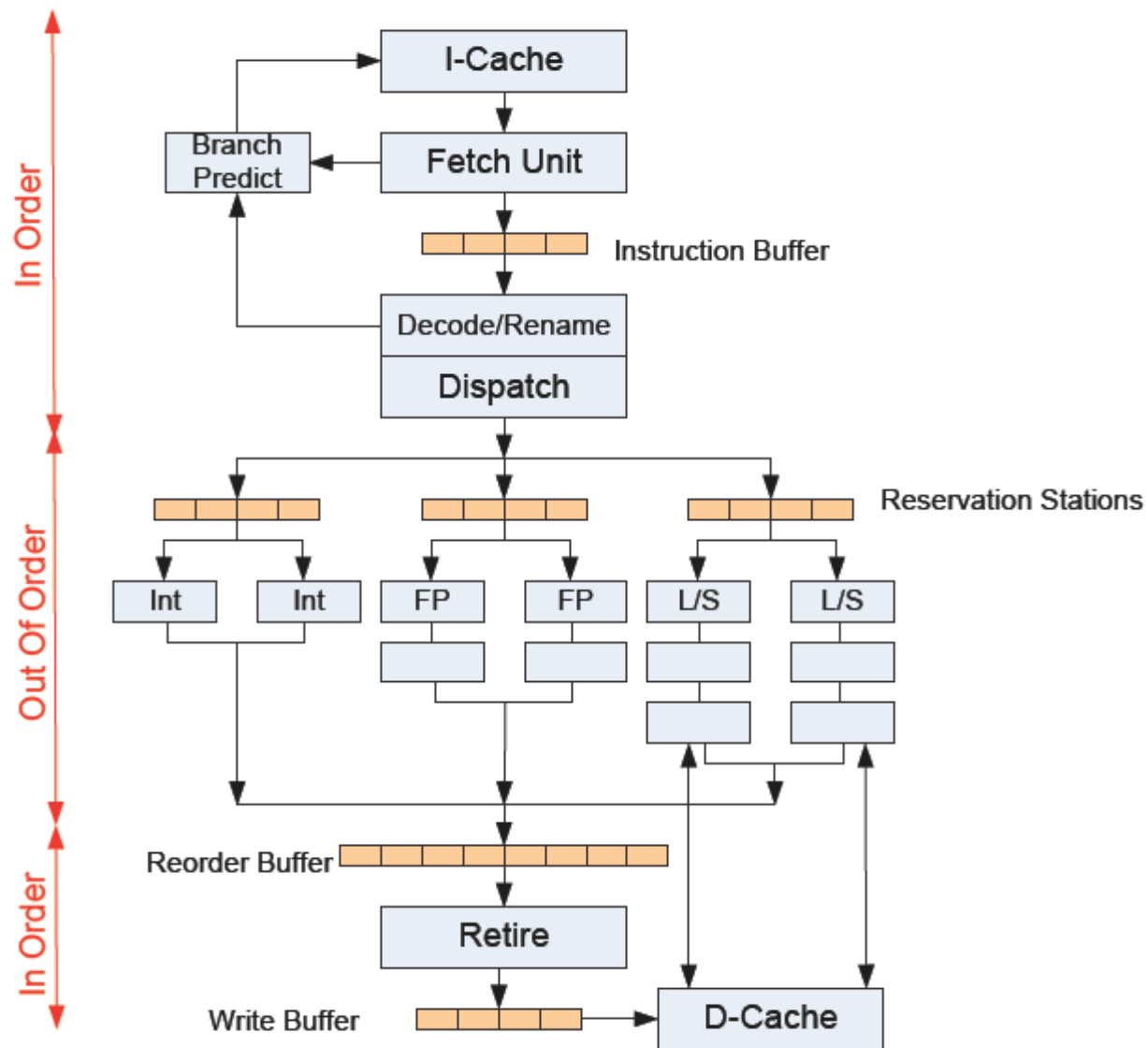
■ Disadvantages

- Parallel execution \rightarrow more dependencies \rightarrow more stalls
 - CPI_{stall} grows due to data and control hazards

流水线技术对处理器性能的影响

- $\text{Execution Time} = \text{IC} \cdot \text{CPI} \cdot \text{CCT}$
 - $\text{CPI} = \text{CPI}_{\text{base}} + \text{CPI}_{\text{stalls}}$
 - Stalls due to data hazards, control hazards, slow memory
- $\text{Power} \cong C \cdot V^2 \cdot F + V_{\text{dd}} \cdot I_{\text{leakage}}$
 - $\text{Energy} = \text{Power} \cdot \text{Execution Time}$
- Comments?

体系结构经典技术示意图：流水线

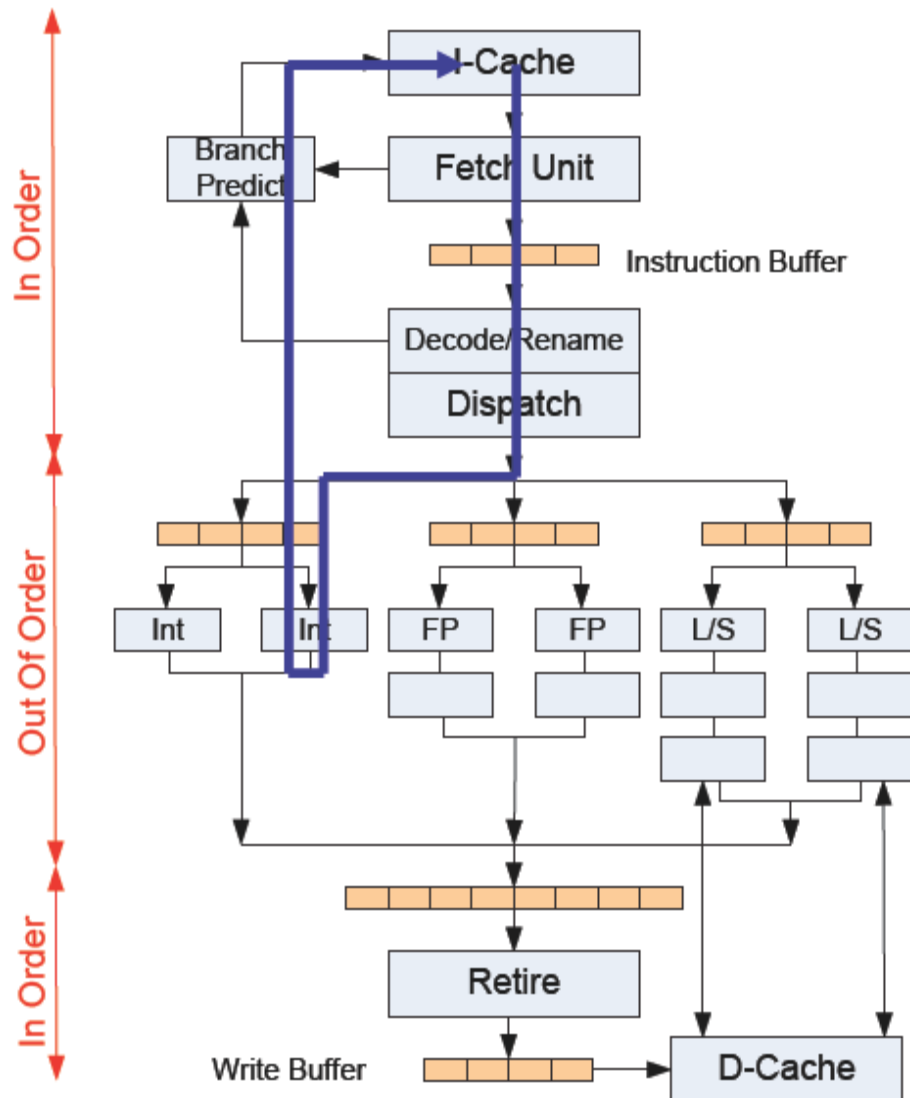


第二讲 走进处理器核设计

■ 处理器核设计技术及其量化分析基础

- 重要性能参数计算
- 现代处理器核设计技术基础
 - Pipelines
 - Branch Prediction
 - Register Renaming
 - Out-of-Order Execution
 - Re-order Buffer
 - Data Parallel Processing:SIMD/Vector Extensions

分支指令的代价 (Branch Penalty)



- **>3 cycles to resolve a branch/jump**
 - Latency of I-cache
 - Decode & execute latency
 - Buffering
- **Cost of branch latency?**
 - Assume 5 cycles to resolve & 4-way superscalar
 - Cost of branch = $5 * 4$ instructions
- **Typical programs:**
 - 1 branch every 4 to 8 instructions

转移猜测: Branch History Table (BHT)

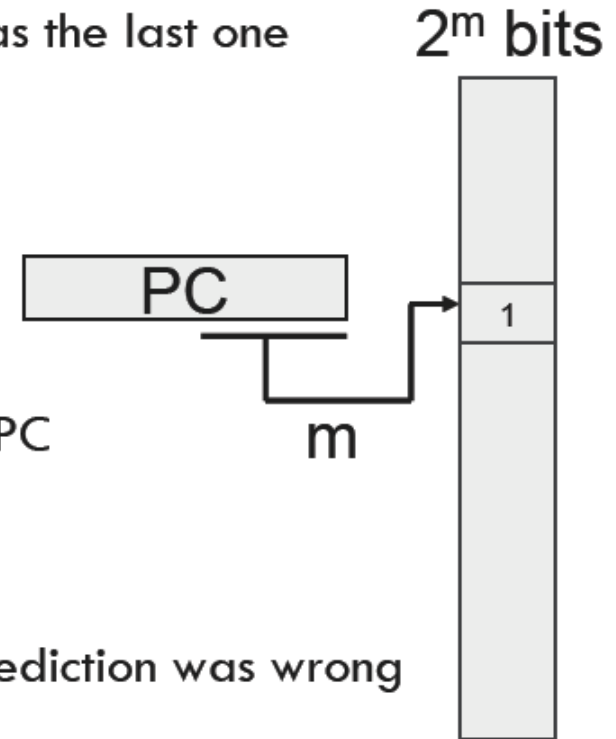
- Basic idea:

- Next branch outcome is likely to be same as the last one

- A $2^m \times 1$ bit table for history

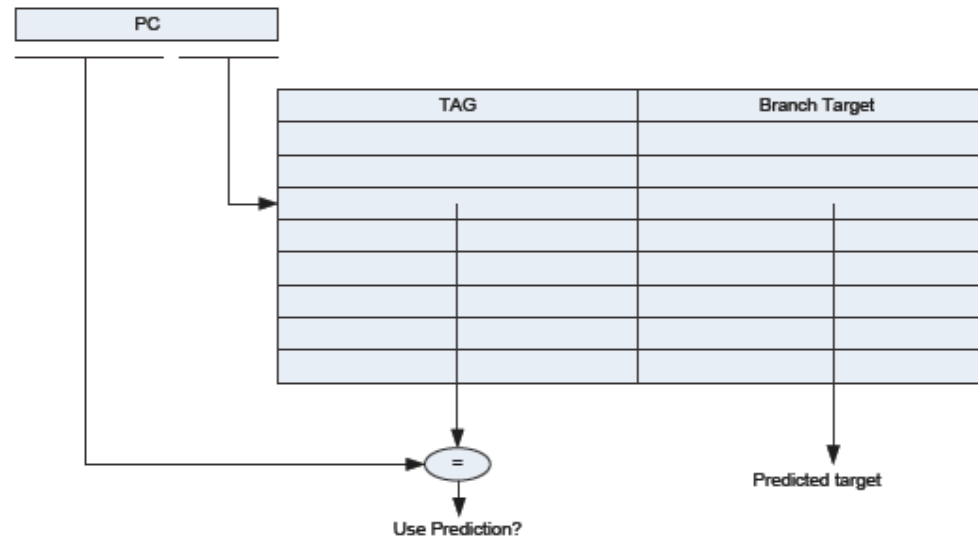
- Algorithm

- Index table with m least significant bits of PC
- If bit==0, predict not taken
- If bit==1, predict taken
- After executing branch, update table if prediction was wrong



- See any shortcomings?
- See any improvements?

带目标地址的转移猜测: Branch Target Buffer (BTB)



■ BTB: A cache for branch targets

- Stores targets for taken branches, jr, function calls
- Reduce size: Don't store prediction for not taken branches
- Algorithm: Access in parallel with instruction cache
 - If hit, use predicted target
 - If miss, use PC+ 16 (assuming 4-way fetch)
 - Update after branch is executed

高级分支预测技术

- Numerous designs and variations
 - Goal: Address shortcomings of BHT & exploit program patterns
- Basic ideas
 - Use >1 b per BHT entry to add hysteresis
 - Use PC & global branch history to address BHT
 - Detect global and local correlation between branches
 - E.g. nested if-then-else statements
 - E.g. short loops
 - Use multiple predictors and select most likely to be correct
 - Capture different patterns with each predictor
 - Measure and use confidence in prediction
 - Avoid executing instructions after difficult to predict branch
 - Neural nets, filtering, separate taken/non-taken streams, ...
- What happens on mispredictions
 - Update prediction tables
 - Flush pipeline & restart from mispredicted target (expensive)

分支预测技术对处理器性能的影响

■ $\text{Execution Time} = \text{IC} \cdot \text{CPI} \cdot \text{CCT}$

■ $\text{CPI} = \text{CPI}_{\text{base}} + \text{CPI}_{\text{stalls}}$

■ Stalls due to data hazards, control hazards, slow memory

■ $\text{Power} \approx C \cdot V^2 \cdot F + V_{\text{dd}} \cdot I_{\text{leakage}}$

■ $\text{Energy} = \text{Power} \cdot \text{Execution Time}$

■ HW Cost?

■ Comments?

第二讲 走进处理器核设计

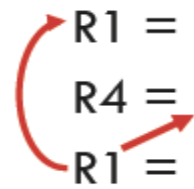
■ 处理器核设计技术及其量化分析基础

- 重要性能参数计算
- 现代处理器核设计技术基础
 - Pipelines
 - Branch Prediction
 - Register Renaming
 - Out-of-Order Execution
 - Re-order Buffer
 - Data Parallel Processing:SIMD/Vector Extensions

处理WAR&WAW相关: 寄存器重命名技术

- WAR and WAW hazards do not represent real data communication

1. $R1 = R2 + R3$
2. $R4 = R1 + R5$
3. $R1 = R6 + R7$



- If we had more registers, we could avoid them completely!

- Register renaming: use more registers than the ~ 32 in the ISA

- Architectural registers mapped to large pool of physical registers
- Give each new “value” produced its own physical register

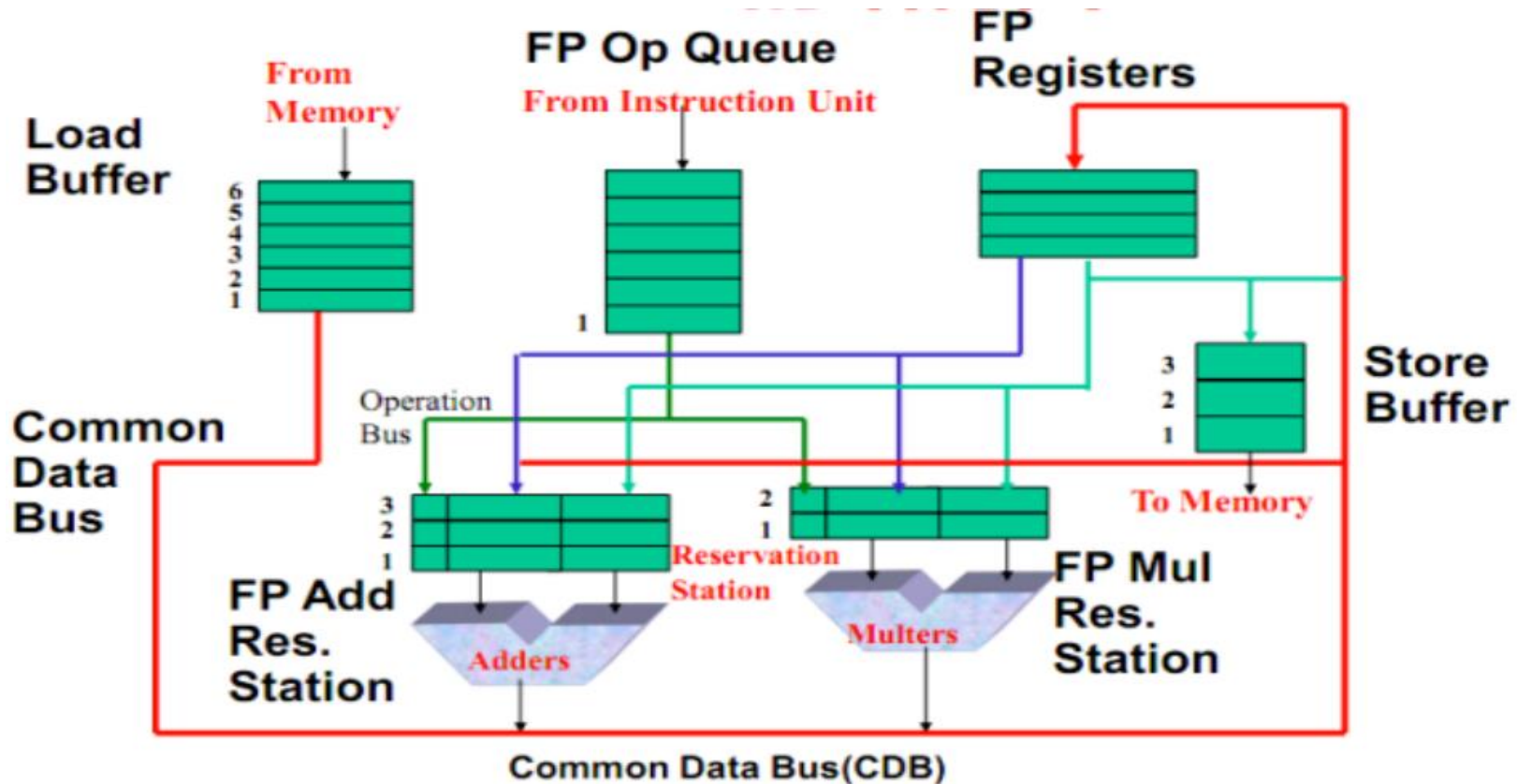
- Before & after renaming

■ $R1 = R2 + R3$	$R1 = R2 + R3$
■ $R4 = R1 + R5$	$R4 = R1 + R5$
■ $R1 = R6 + R7$	$R33 = R6 + R7$
■ $R6 = R1 + R3$	$R34 = R33 + R3$

- Which key technique are we using here?

Tomasulo algorithm

- Robert Tomasulo(1934 –2008)
- 计算机科学家。毕业于Manhattan College，1956年加入IBM，1966年在研制IBM 360大型机时发明Tomasulo算法。1997年因该算法获得Eckert–Mauchly Award。
- 逻辑寄存器与物理寄存器



乱序执行技术

- **In-order execution:** Instruction dispatched to a functional unit when
 - All older instructions have been dispatched
 - All operands are available & FU available
- **Out-of-order execution:** Instruction dispatched when
 - All operands are available & FU available
- **Essentially, OOO execution recreates data-flow order**
- **Implementation**
 - Reservation stations or instruction window
 - Keep track when operands become available

访存乱序该怎么处理？

■ When can a load read from the cache?

- Option 1: When its address is available & all older stores done
- Option 2: When its address is available, all older stores have address available, and no RAW dependency
- Option 3: When its address is available
 - Speculate no dependency with older stores, must check later

■ When can a store write to the cache?

- It must have its address & data
- All previous instructions must be exception free
- It must be exception free
- All previous loads have executed or have address
 - No dependency

■ Implementation with load/store buffers with associative search

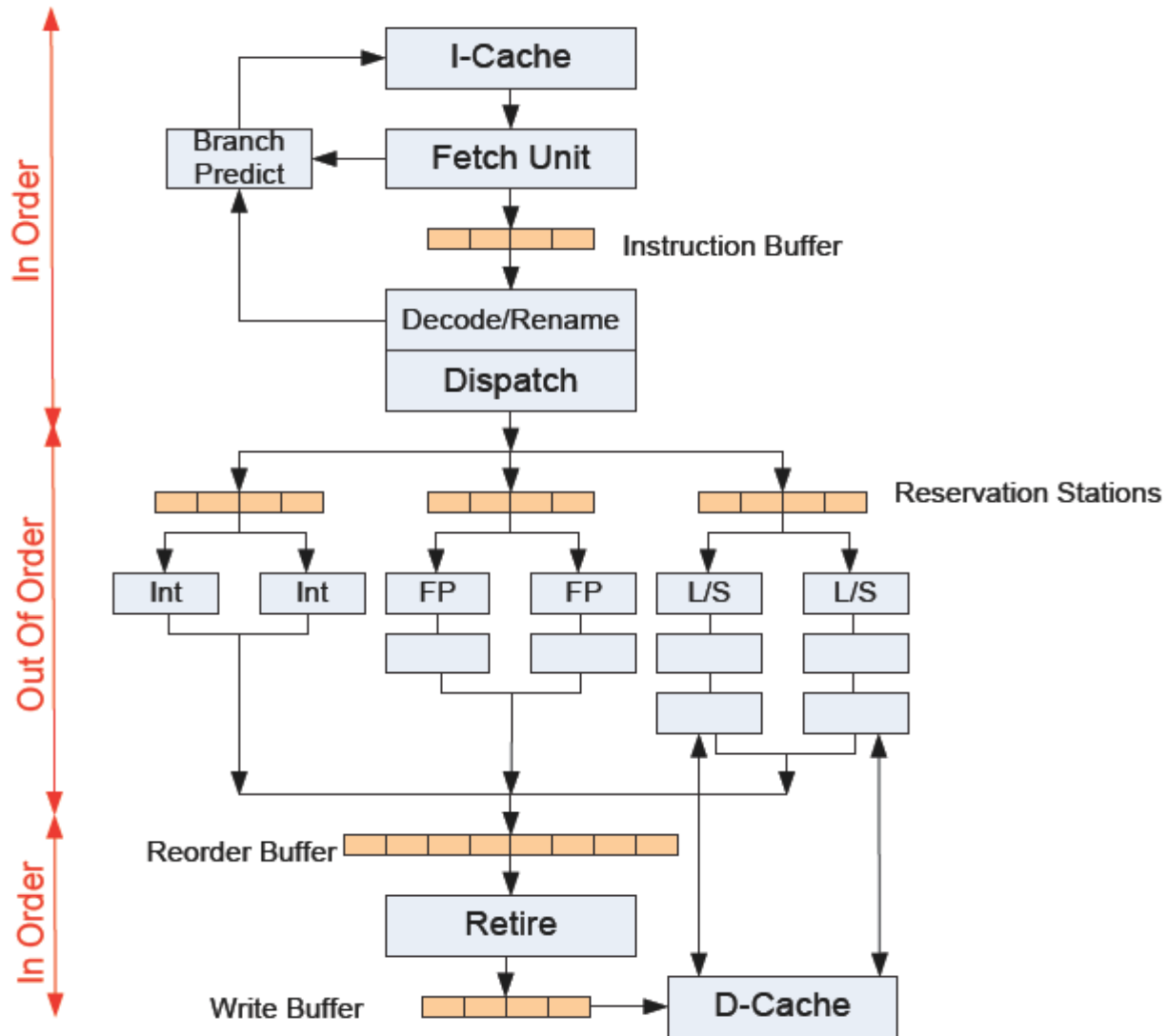
实现精确例外：Re-order Buffer技术

- Precise exceptions: exceptions must occur in same order as in unpipelined, single-cycle processor
 - Older instruction first, no partial execution of younger instructions
- Re-order buffer: A FIFO buffer for recapturing order
 - Space allocated during instruction decode, in-order
 - Result updated when execution completes, out-of-order
 - Result written to registers or write-buffer in-order
 - Older instruction first
 - If older instruction not done, stall
 - If older instruction has exception, flush buffer to eliminate results of incorrectly executed instructions

乱序执行技术对处理器性能的影响

- $\text{Execution Time} = \text{IC} \cdot \text{CPI} \cdot \text{CCT}$
 - $\text{CPI} = \text{CPI}_{\text{base}} + \text{CPI}_{\text{stalls}}$
 - Stalls due to data hazards, control hazards, slow memory
- $\text{Power} \approx C \cdot V^2 \cdot F + V_{\text{dd}} \cdot I_{\text{leakage}}$
 - $\text{Energy} = \text{Power} \cdot \text{Execution Time}$
- HW cost?
- Comments?

超标量乱序执行处理器示意图

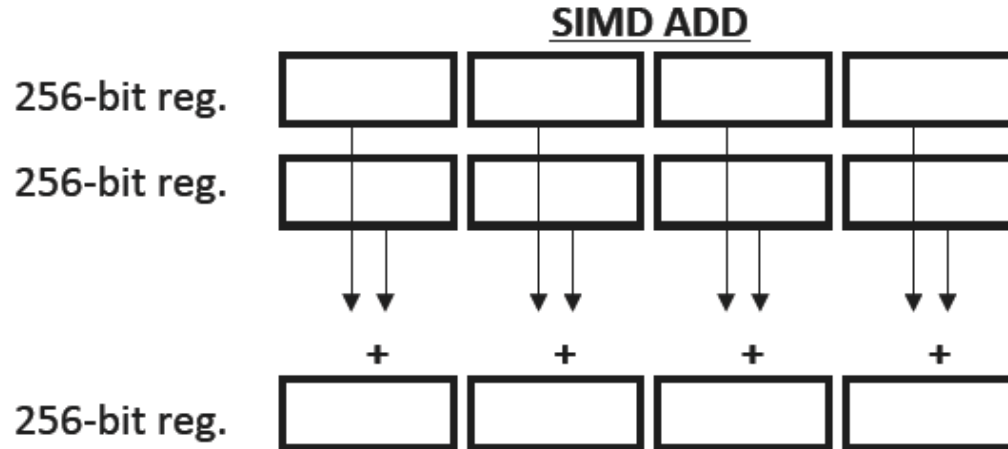


第二讲 走进处理器核设计

■ 处理器核设计技术及其量化分析基础

- 重要性能参数计算
- 现代处理器核设计技术基础
 - Pipelines
 - Branch Prediction
 - Register Renaming
 - Out-of-Order Execution
 - Re-order Buffer
 - Data Parallel Processing:SIMD/Vector Extensions

处理器中的数据并行技术:SIMD/Vector Extensions



- **Goal: improve compute throughput for data parallel codes**
 - Without increasing processor width
- **SIMD/Vector: each instruction defines multiple identical ops**
 - New registers to store vectors of operands
 - New ALUs to processor vectors of operands
- **Examples: MMX, SSE-{2,3,4}, AVX**
 - From 32b to 256b vectors

SIMD/Vector技术对处理器性能的影响

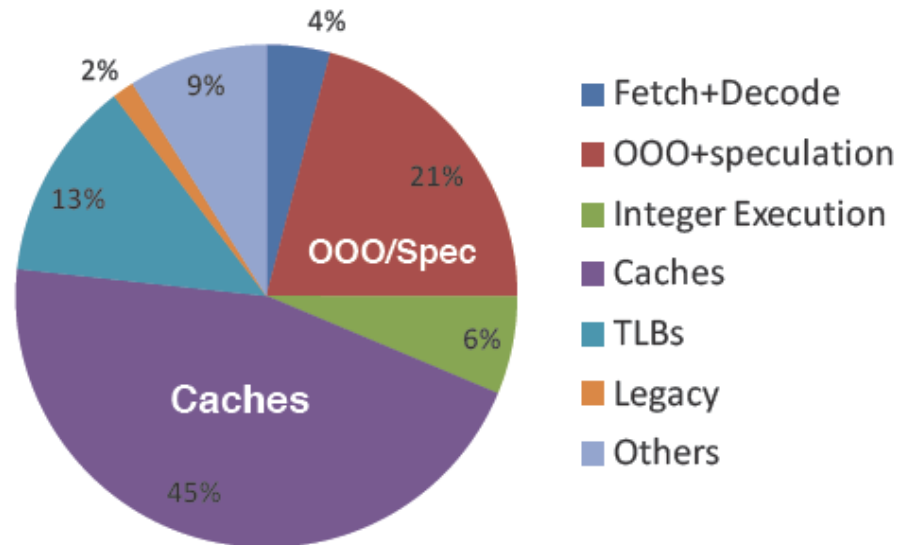
- $\text{Execution Time} = \text{IC} \cdot \text{CPI} \cdot \text{CCT}$
 - $\text{CPI} = \text{CPI}_{\text{base}} + \text{CPI}_{\text{stalls}}$
 - Stalls due to data hazards, control hazards, slow memory
- $\text{Power} \approx C \cdot V^2 \cdot F + V_{\text{dd}} \cdot I_{\text{leakage}}$
 - $\text{Energy} = \text{Power} \cdot \text{Execution Time}$
- HW cost?
- Comments?

现代处理器核设计挑战

- Power consumption
 - Gets worse with higher clock & more OOO logic
- Design complexity
 - Grows exponentially with issue width
- Limited ILP
- Clock frequency: getting close to pipelining limits
 - Clocking overheads, CPI degradation
- Branch prediction & memory latency
 - Limit the practical benefits of out-of-order execution
- Increasingly difficult to scale single-processor architectures → shift to multi-core chips

Power Breakdown of Processor

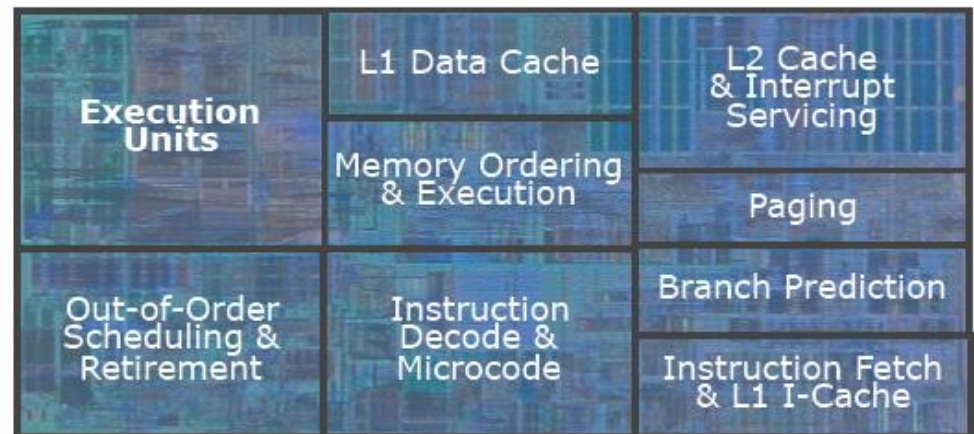
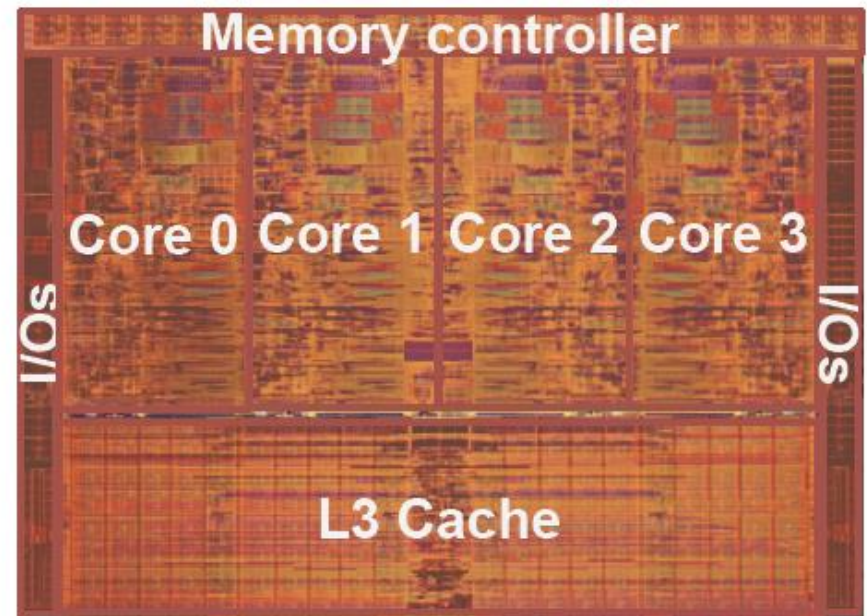
[Sodani, 2011]



- Implications?
- Can we increase performance without increasing cost of instruction fetch, decode, and scheduling?

走向多核: Intel Core i7 (Nehalem)

- 4 cores/chip
- 16 pipeline stages, ~3GHz
- 4-wide superscalar
- Out of order, 128-entry reorder buffer
- 2-level branch predictors
- SSE-4 SIMD extensions
- Caches
 - L1: 32KB I + 32KB D
 - L2: 256KB
 - L3: 8MB, shared



致谢:

本讲部分内容参考了M.I.T. Daniel Sanchez教授的课程讲义，特此感谢。