Parallel Programming Principle and Practice

Lecture 2 — Single Processor System

Outline

- Basic components of computer
  - CPU
  - Controller
  - Memory
- Instruction level parallelism
  - Pipelining
  - Superscalar
  - Stream SIMD extension (SSE), advanced vector extension (AVX)
  - ILP is managed by compiler and hardware
  - Complier and hardware only have limited capability for optimization to many problems
- Memory hierarchy
  - Registers
  - Main memory
  - Cache
  - Why Cache?

CPU

Central processing unit (CPU: 中央处理器)

- Called a central processor, main processor or just processor
- Electronic circuitry that executes instructions comprising a computer program
- Performs basic arithmetic, logic, controlling, and input/output (I/O) operations specified by the instructions in the program

Controller (控制器）

- Control Unit
  - Reads an instruction from memory (at PC)
  - Interprets the instruction
  - Generates signals that tell the other components what to do

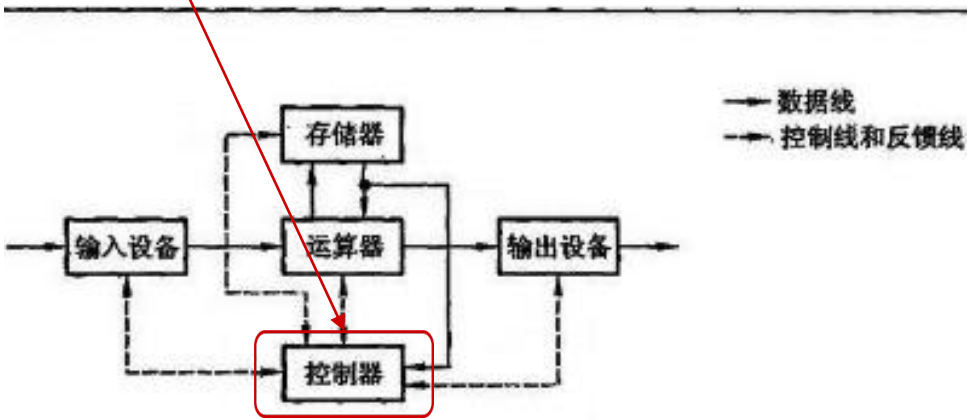


图 1.7　典型的冯·诺依曼计算机结构框图

Memory（存储器）

- One of the important components of a computer, also known as internal memory（内存） and main memory （主存）
- Used to temporarily store operational data in the CPU and data exchanged with external memory （外存） such as hard disks （硬盘）
- The bridge between the external memory and the CPU.

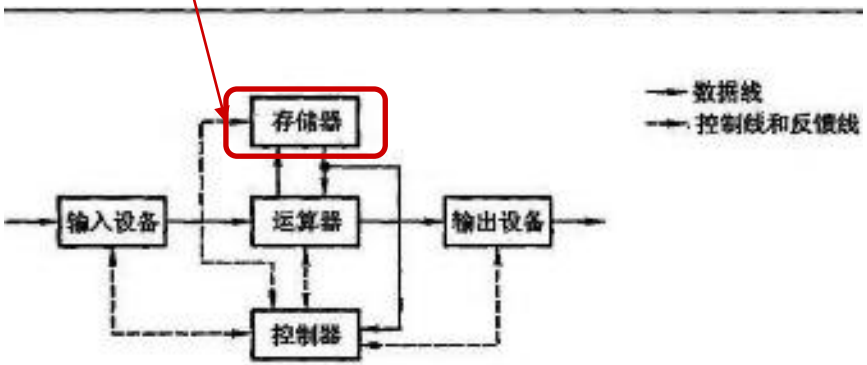


图 1.7 典型的冯·诺依曼计算机结构框图

Single Processor System

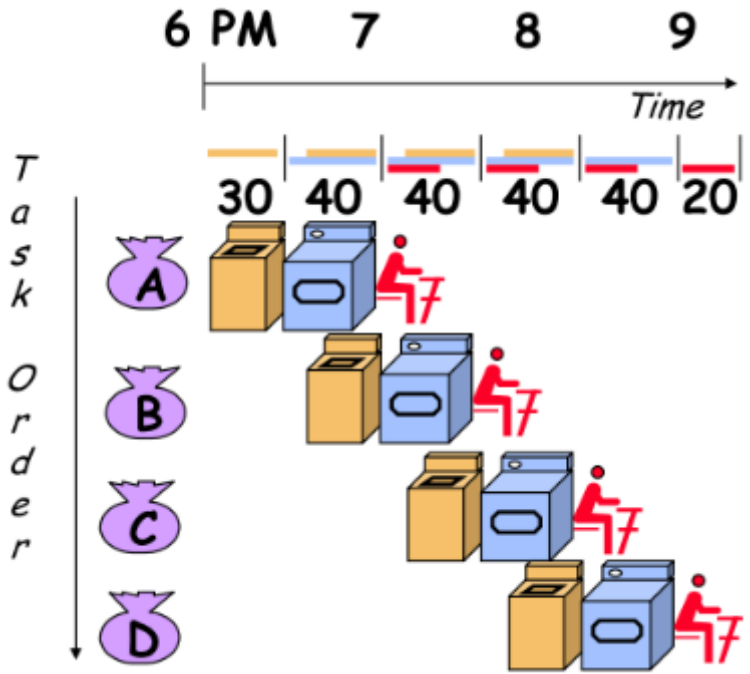
Instruction level parallelism

SINGLE PROCESSOR ALSO A PARALLEL PROCESSING ELEMENT

Pipelining (流水线)

David Patterson's Laundry example: 4 people doing laundry

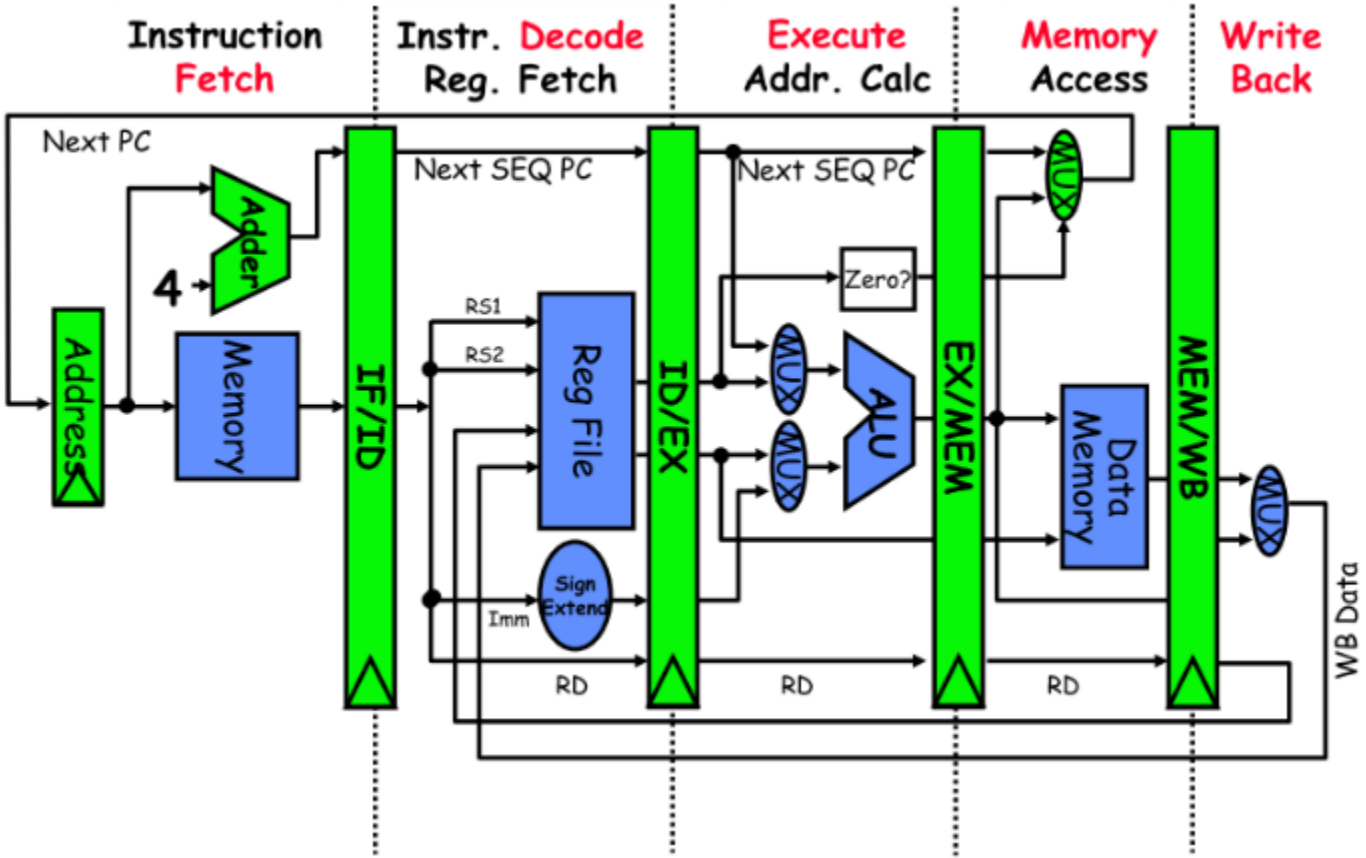
wash (30 min) + dry (40 min) + fold (20 min) = 90 min Latency



- In this example
  - Sequential execution takes 4 * 90min = 6 hours
  - Pipelined execution takes 30+4*40+20 = 3.5 hours
- Bandwidth = loads/hour
  - BW = 4/6 l/h w/o pipelining
  - BW = 4/3.5 l/h w pipelining
  - BW <= 1.5 l/h w pipelining, more total loads
- Pipelining helps bandwidth (带宽) but not latency (延迟，90 min)
- Bandwidth limited by slowest pipeline stage
- Potential speedup = Number of pipe stages

5 Steps of MIPS Pipeline

- Instruction fetch
  取指
- Instruction decode
  译码
  register fetch
  读取通用寄存器组
- Execute 执行
- Memory access
  访存
- Write back
  写回



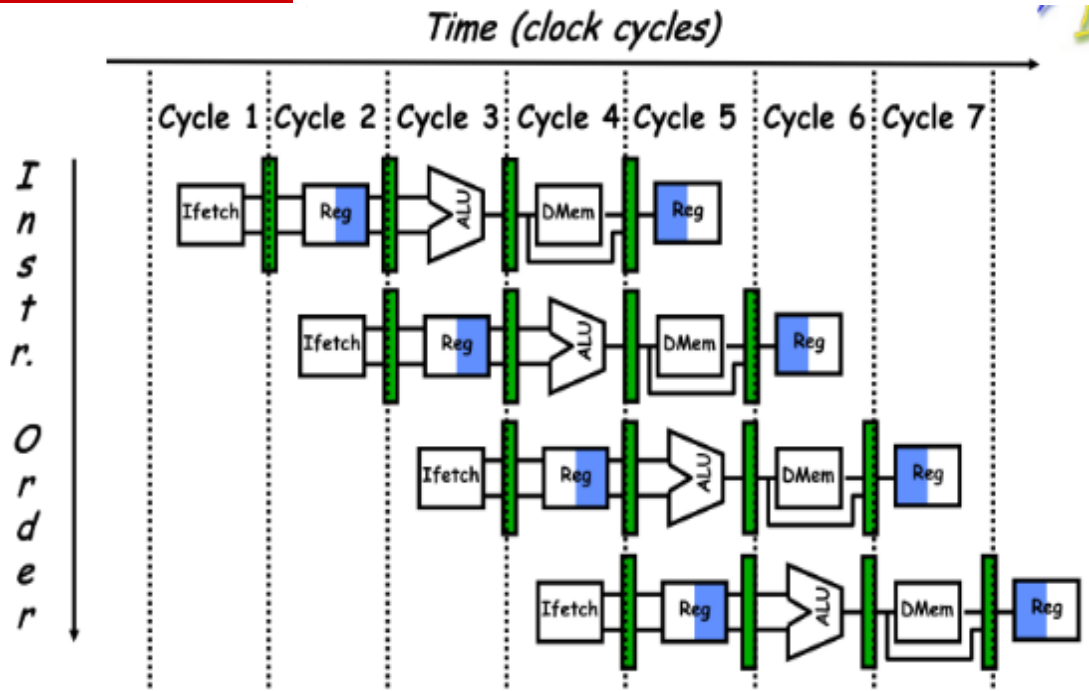
Visualizing Pipeline

- In ideal case:

  CPI (cycles/instruction) = 1!

  Cycle:时钟频率

  - On average, put one instruction into pipeline, get one out



Limits to Pipelining

- Overhead (开销) prevents arbitrary division
  - Cost of latches (锁存器) (between stages) limits what can do within stage
  - Sets minimum amount of work/stage
- Hazards (冲突) prevent next instruction from executing during its designated clock cycle
  - Structural hazards (结构冲突): Attempt to use the same hardware to do two different things at once
  - Data hazards (数据冲突): Instruction depends on result of prior instruction still in the pipeline
  - Control hazards (控制冲突): Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps)

Superscalar（多计算单元、超标量）

- A superscalar machine executes multiple independent instructions in parallel (Launch more than one instruction/cycle)
  - They are pipelined as well
  - In ideal case, CPI < 1 (Launch more than one instruction/cycle)
- “Common” instructions (arithmetic, load/store, conditional branch) can be executed independently
- Equally applicable to RISC (精简指令集) & CISC (复杂指令集), but more straightforward in RISC machines
- The order of execution is usually assisted by the compiler
- Superscalar increases occurrence of hazards
  - More conflicting instructions/cycle

Source:https://en.wikipedia.org/wiki/Central_processing_unit

Superscalar

- View of Superscalar Execution

Instruction dispatch

指令分派

Instruction issue

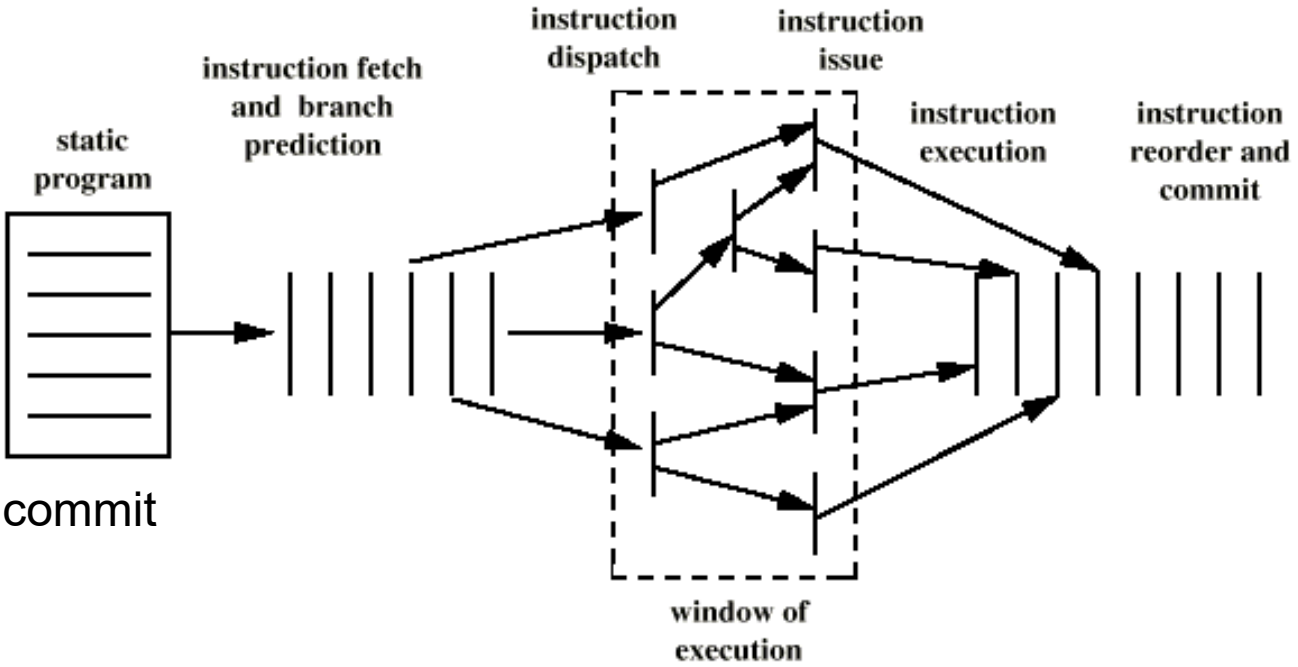
指令发送

Instruction execution

指令执行

Instruction reorder and commit

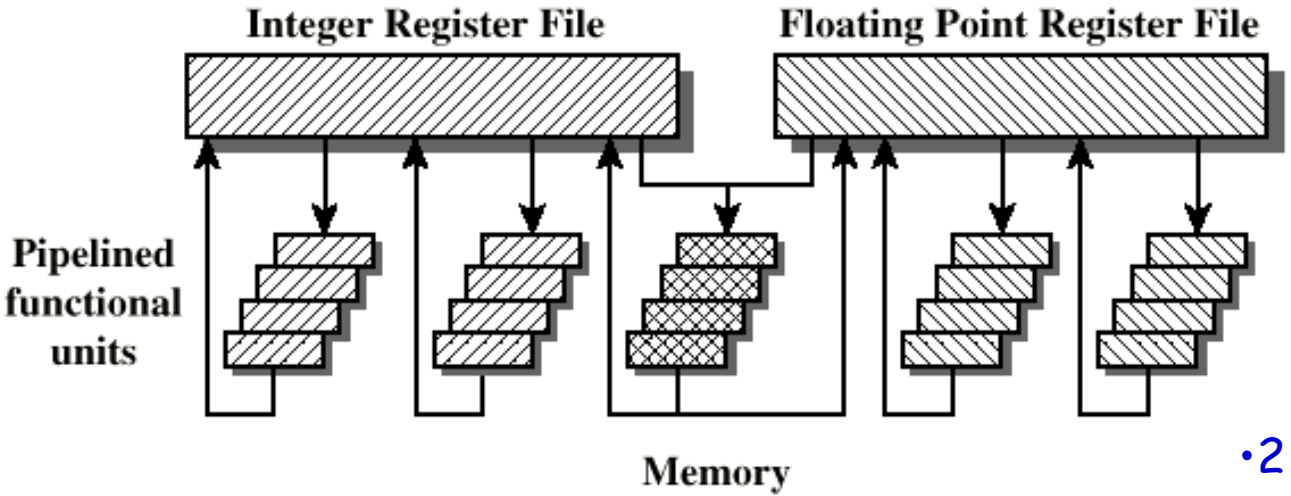
指令重排和提交



Source:https://en.wikipedia.org/wiki/Central_processing_unit

Superscalar

- Example of Superscalar Organization

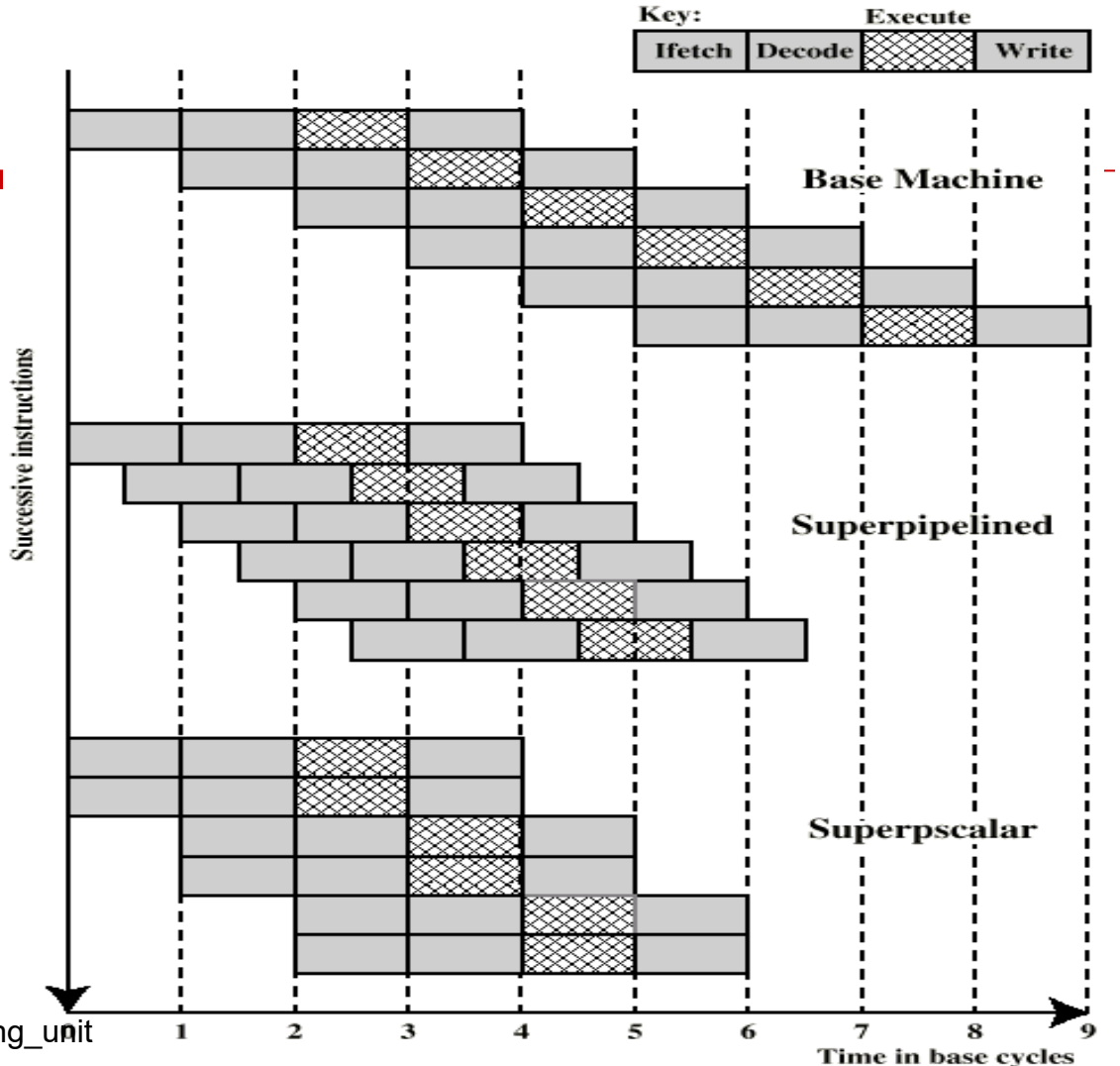


- 2 Integer ALU pipelines,
- 2 FP ALU pipelines,
- 1 memory pipeline

Source:https://en.wikipedia.org/wiki/Central_processing_unit

Superscalar

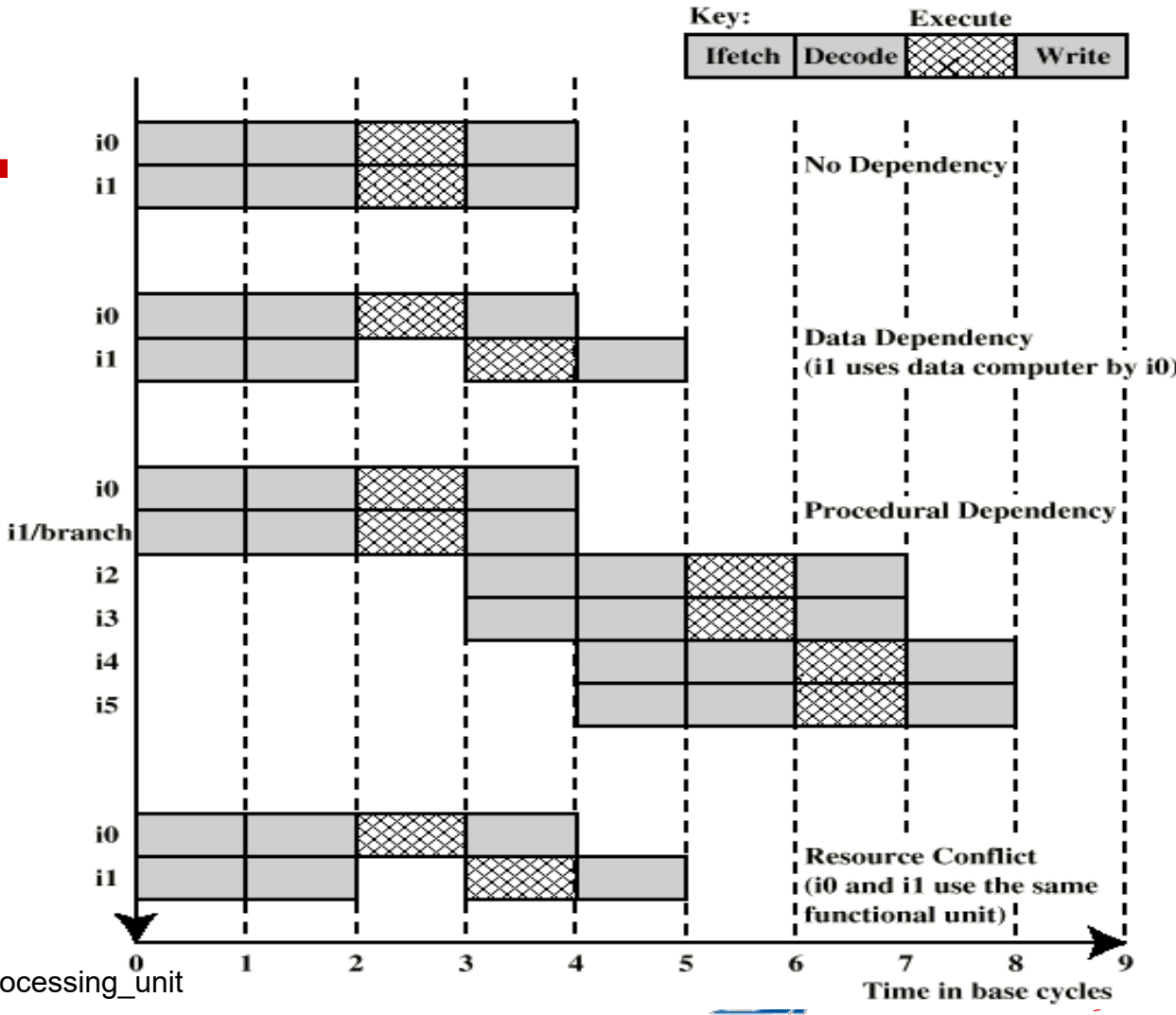
- Superscalar v Superpipelined



Source:https://en.wikipedia.org/wiki/Central_processing_unit

Superscalar

- Effect of Dependencies on Superscalar Operation
  - Superscalar operation is double impacted by a stall (阻塞)
  - CISC machines typically have different length instructions and need to be at least partially decoded before the next can be fetched – not good for superscalar operation



Source:https://en.wikipedia.org/wiki/Central_processing_unit

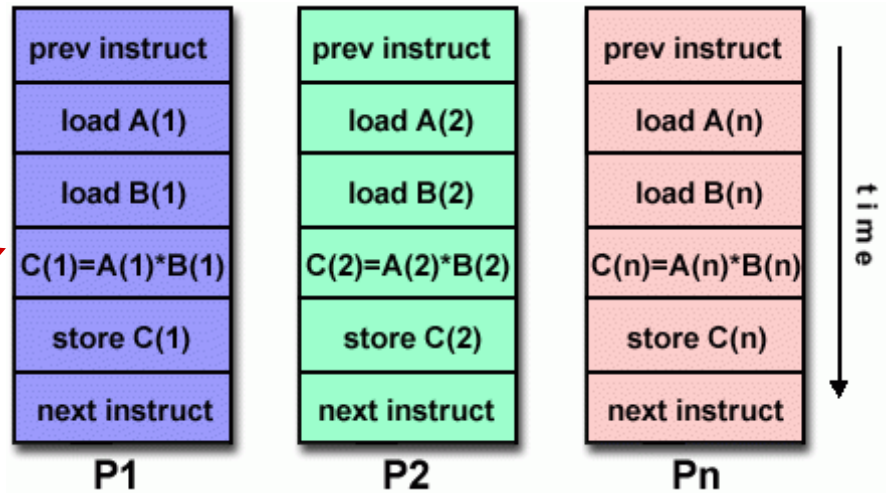
Limitations of Superscalar

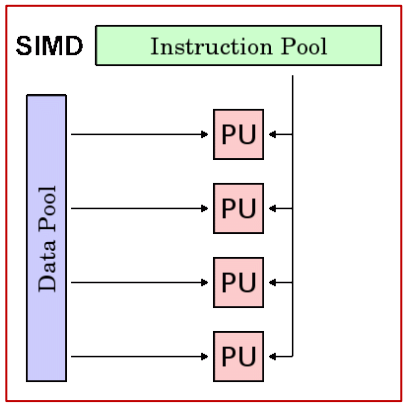
- Dependent upon
  - Instruction level parallelism possible
  - Compiler based optimization
  - Hardware support
- Limited by
  - Data dependency
  - Procedural dependency
  - Resource conflicts

Source:https://en.wikipedia.org/wiki/Central_processing_unit

SIMD: Single Instruction, Multiple Data

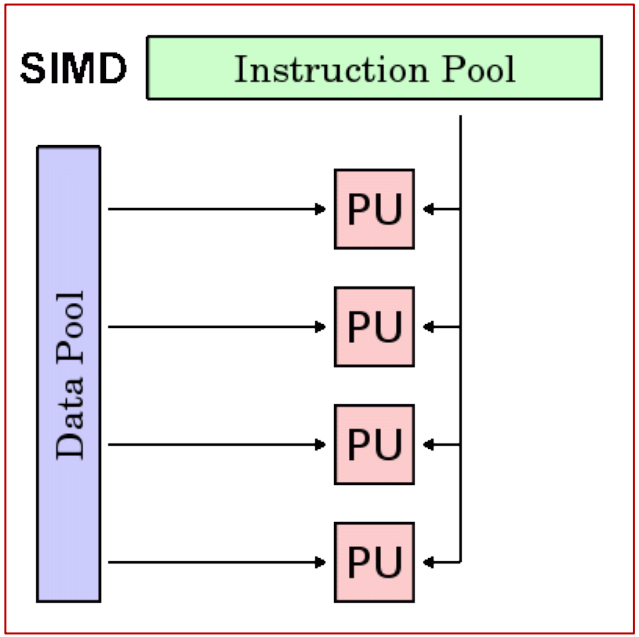
- A type of parallel computer
  - Single Instruction
    - ✓ All processing units execute the same instruction at any given clock cycle
  - Multiple Data
    - ✓ Each processing unit can operate on a different data element





SIMD: Single Instruction, Multiple Data

- A type of parallel computer
  - Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
  - Synchronous (同步) (lockstep) and deterministic execution
  - Two varieties
    - ✓ Processor Arrays
    - ✓ Vector Pipelines

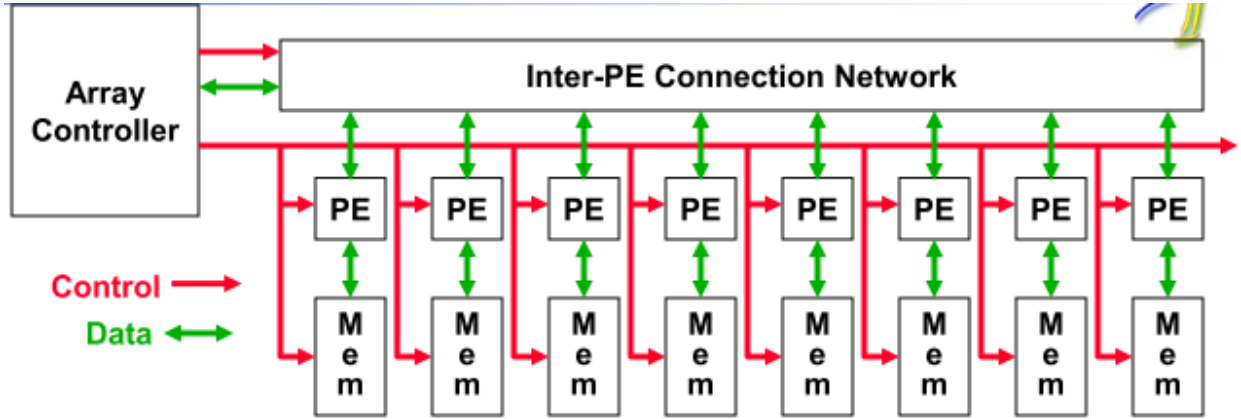


SIMD: Single Instruction, Multiple Data

- A type of parallel computer
  - Examples
    - ✓ Processor Arrays: Thinking Machines CM-2, MasPar MP-1 & MP-2, ILLIAC IV
    - ✓ Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10
    - ✓ Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.

SIMD Architecture

- Central controller broadcasts instructions to multiple processing elements (PEs)
  - Only requires one controller for whole array
  - Only requires storage for one copy of program
  - All computations are fully synchronized
- Recent return to popularity
  - GPU (Graphics Processing Units) have SIMD properties
  - However, also multicore behavior, so mix of SIMD and MIMD (more later)



SSE: Streaming SIMD Extensions

- A single instruction, multiple data (SIMD) instruction set extension to the x86 architecture
- Designed by Intel and introduced in 1999 in their Pentium III series of Central processing units (CPUs) shortly after the appearance of Advanced Micro Devices (AMD's) 3DNow!
- Contains 70 new instructions, most of which work on single precision floating-point data.
- Greatly increase performance when exactly the same operations are to be performed on multiple data objects.
- Typical applications are digital signal processing and graphics processing.

Source: https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions

AVX: Advanced Vector Extensions

- Doubles the width to 256 bits
- Extendible to 512 and 1024 bits for future generations
- Operands must be consecutive and aligned memory locations
- Extensions to the x86 instruction set architecture for microprocessors from Intel and Advanced Micro Devices (AMD).
- Proposed by Intel in March 2008 and first supported by Intel with the Sandy Bridge processor shipping in Q1 2011 and later by AMD with the Bulldozer processor shipping in Q3 2011.
- Provides new features, new instructions and a new coding scheme.

Source: https://en.wikipedia.org/wiki/Advanced_Vector_Extensions

ILP

- ILP: Instruction Level Parallelism
  - Managed by compiler (编译器) and hardware
  - ILP allows the compiler and the processor to overlap the execution of multiple instructions or even to change the order in which instructions are executed
- Examples of ILP techniques
  - Pipelining: Overlapping individual parts of instructions
  - Superscalar execution: Do multiple things at same time
  - VLIW (超长指令字): Let compiler specify which operations can run in parallel
  - Vector Processing (向量处理): Specify groups of similar (independent) operations
  - Out of Order Execution (OOO，乱序执行): Allow long operations to happen

Modern ILP

- Dynamically scheduled, out-of-order execution
  - Current microprocessors fetch (取) 6-8 instructions per cycle
  - Pipelines are 10s of cycles deep → many overlapped instructions in execution at once, although work often discarded (放弃，变更)
- What happens
  - Grab a bunch of instructions → determine all their dependences → eliminate dep's wherever possible →throw them all into the execution unit → let each one move forward as its dependences are resolved
  - Appears as if executed sequentially

Modern ILP（Cont'd）

- Dealing with Hazards: May need to *guess!*
  - Called "Speculative Execution (预测执行)"
  - Speculate on Branch results, Dependencies, even Values!
    - If correct, don't need to stall (阻塞) for result → yields performance
    - If not correct, waste time *and power*
  - Must be able to UNDO a result if guess is wrong
  - Problem: accuracy of guesses decreases with number of simultaneous instructions in pipeline
- Huge complexity
  - Complexity of many components scales as n^2 (issue width)
  - Power consumption big problem

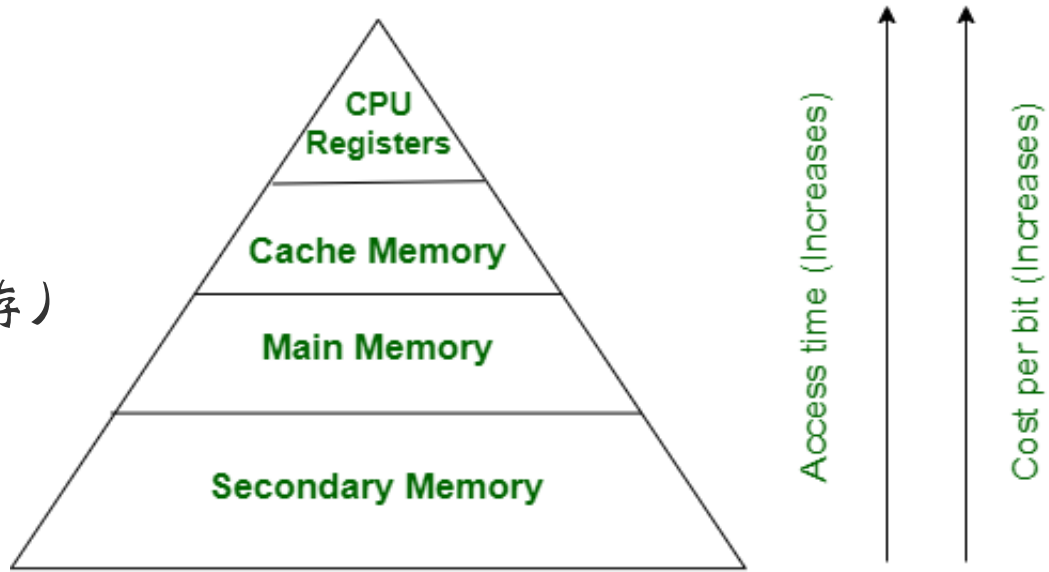
Single professor system

Memory hierarchy（存储器层次结构）

Memory hierarchy

- Computer Memories store data and instruction.
- Memory system can be divided into 4 categories
  - Registers（寄存器）
  - Caches（快速缓存）
  - Main memory（主存）
  - External memory/storage（外存）

They can be represented in an hierarchical form as:



https://www.geeksforgeeks.org/difference-between-primary-and-secondary-memory/

Registers

- Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU.
- Registers are managed by compiler.
- Processor registers
  - The registers used by the CPU are often termed as Processor registers.
  - A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).
  - The computer needs processor registers for manipulating data and a register for holding a memory address.
  - The register holding the memory location is used to calculate the address of the next instruction after the execution of the current instruction is completed.

https://www.javatpoint.com/computer-registers

Registers

- Following is the list of some of the most common registers used in a basic computer

Register	Symbol	Number of bits	Function
Data register	DR	16	Holds memory operand
Address register	AR	12	Holds address for the memory
Accumulator	AC	16	Processor register
Instruction register	IR	16	Holds instruction code
Program counter	PC	12	Holds address of the instruction
Temporary register	TR	16	Holds temporary data
Input register	INPR	8	Carries input character
Output register	OUTR	8	Carries output character

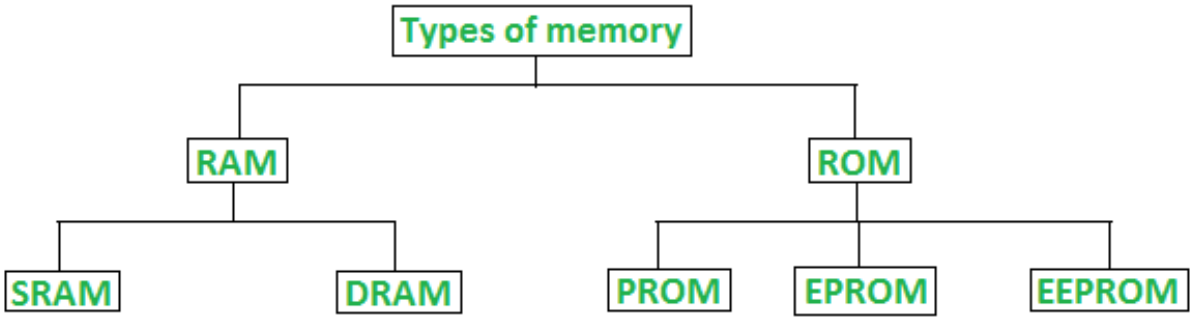
Main memory

- Managed by OS software
- Name
  - Main memory refers to physical memory that is internal to the computer.
  - The word main is used to distinguish it from external mass storage devices such as disk drives.
  - Other terms used to mean main memory include RAM and primary storage.
- execute
  - The computer can manipulate only data that is in main memory.
  - Therefore, every program you execute and every file you access must be copied from a storage device into main memory.
  - The amount of main memory on a computer is crucial because it determines how many programs can be executed at one time and how much data can be readily available to a program.

Main memory

- Primary memory can be broadly classified into two parts

1. Read-Only Memory (ROM)
2. Random Access Memory (RAM)



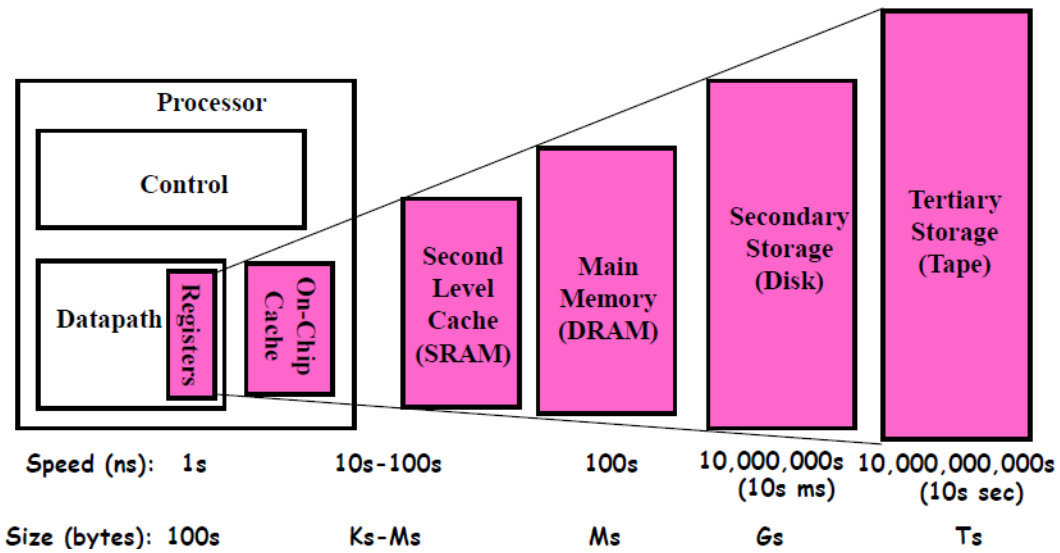
Classification of computer memory

Cache

- Caches hold multi-word cache lines
- Caches are managed by hardware
- In computing, a cache is a hardware or software component that stores data so that future requests for that data can be served faster.
  - The data stored in a cache might be the result of an earlier computation or a copy of data stored elsewhere.
  - A cache hit occurs when the requested data can be found in a cache, while a cache miss occurs when it cannot.
  - Cache hits are served by reading data from the cache, which is faster than recomputing a result or reading from a slower data store; thus, the more requests that can be served from the cache, the faster the system performs.

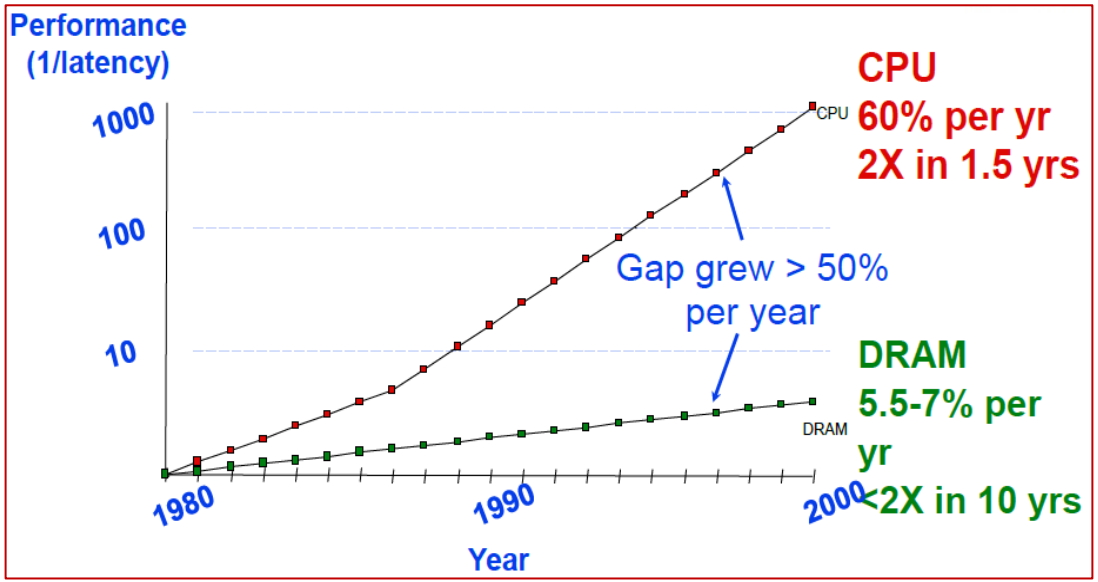
Why Cache?

- Take advantage of the principle of locality to
  - Present as much memory as in the cheapest technology
  - Provide access at speed offered by the fastest technology



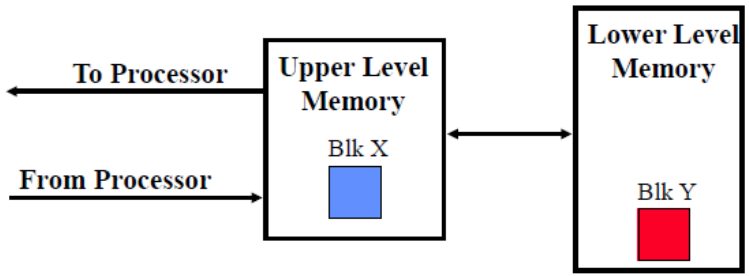
Why Cache?

- Limiting Force: Memory Wall
- How do architects address this gap?
  - Put small, fast “cache” memories between CPU and DRAM (Dynamic Random Access Memory).
  - Create a “memory hierarchy”



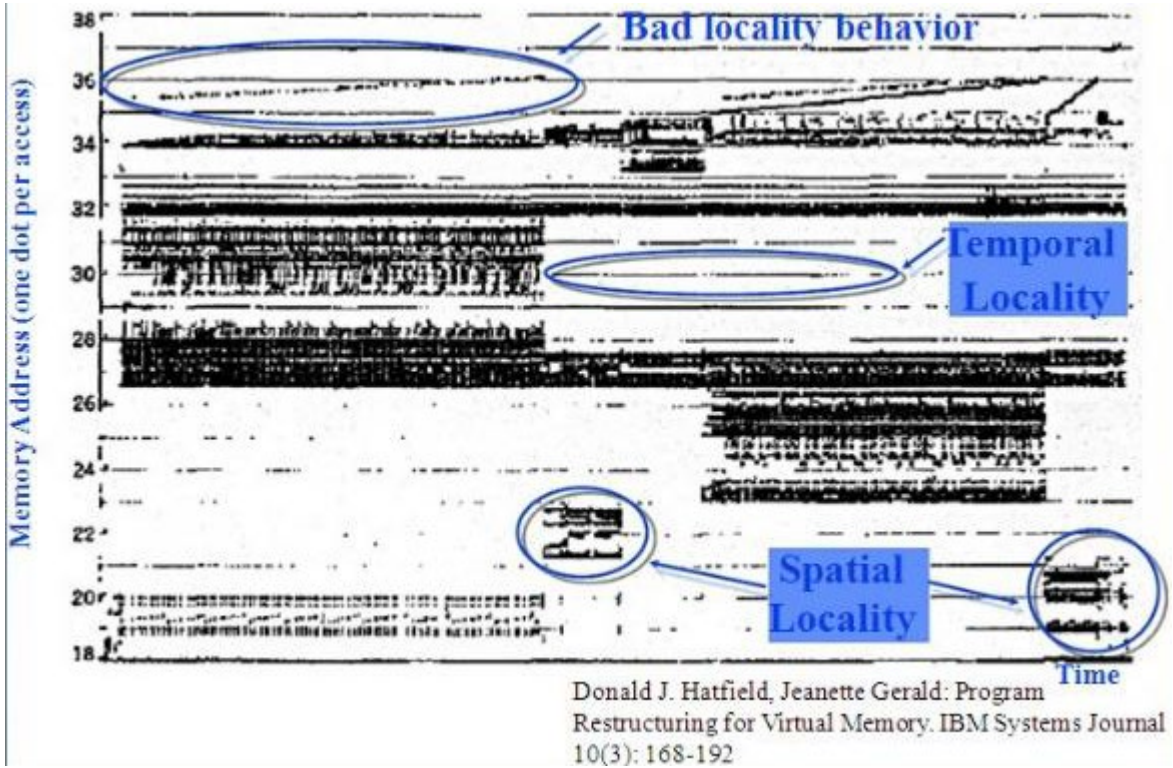
Why Cache?

- Hit: data appears in some blocks in the upper level (example: Block X)
  - Hit Rate: The fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieve from a block in the lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level + Time to deliver the block to the processor
- Hit Time << Miss Penalty (500 instructions on 21264!)



Why Cache?

- Programs with locality cache well



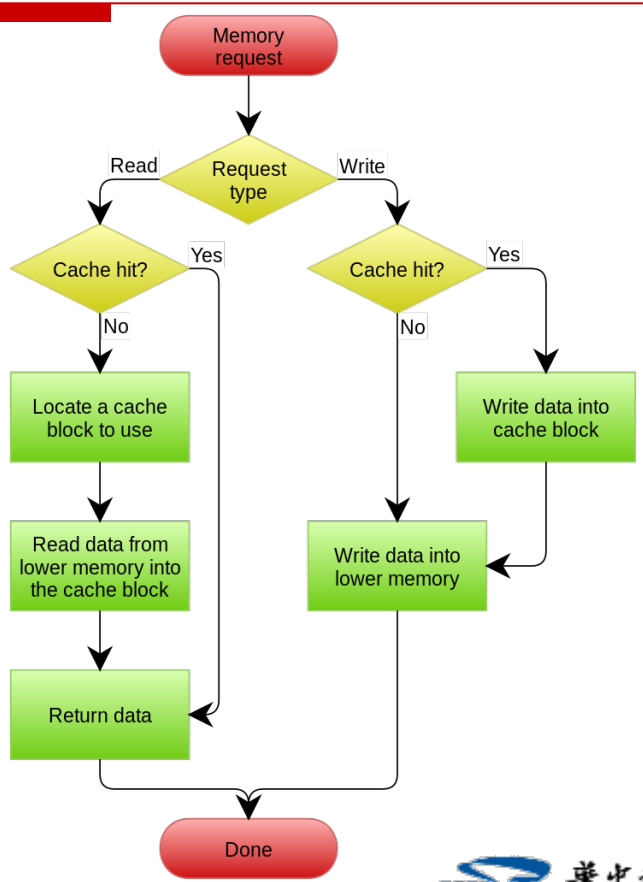
Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192

Why Cache?

- Principle of Locality
  - Program access a relatively small portion of the address space at any instant of time
- Two Different Types of Locality
  - *Temporal Locality* (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - *Spatial Locality* (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Last 25 years, Hardware relied on locality for speed

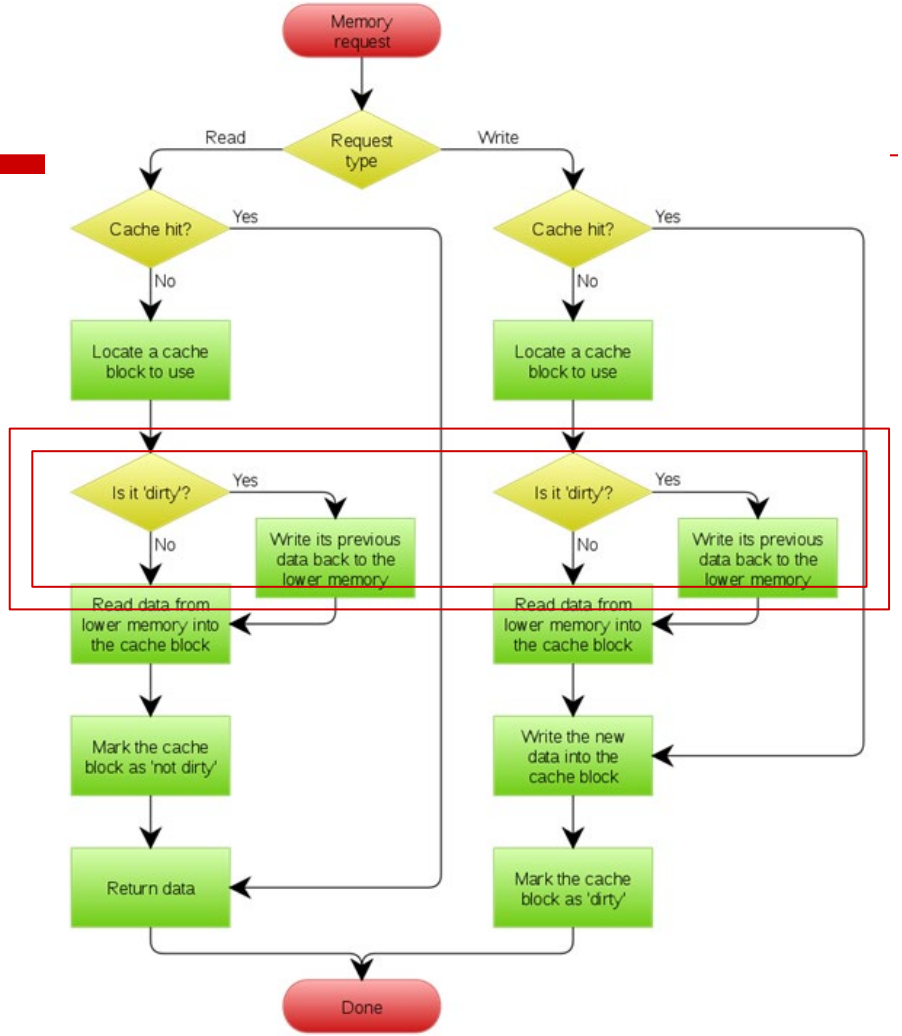
Cache algorithm

- A write-through cache (写穿模式) with no-write allocation
  - 该模式下，CPU对主存写数据时，不经过cache直接写到内存
  - 此时对于写的实现比较简单，如果系统只用写穿模式的话，cache则变成了读缓存了



Cache algorithm

- A write-back cache with write allocation
- CPU写入数据时，不直接将数据写入内存，而是写入cache，当cache数据被替换出去时才写回主存
- 所有数据都要写回么？
- 脏数据：
  - cache中是数据和主存中的数据是否不一致
  - cache中每一块增加了一个记录信息位



Conclusion

- Basic components of computer
- Instruction level parallelism
  - Pipelining
  - Superscalar
  - ILP is managed by compiler and hardware
- Memory hierarchy
  - Registers
  - Main memory
  - Cache
  - Why Cache?