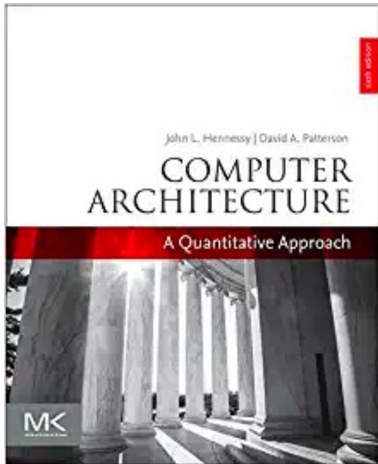


Chapter 1

Fundamentals of Quantitative Design and Analysis

Part 2: Trends, Constraints, and Wisdom

"I think it's fair to say that personal computers have become the most empowering tool we've ever created. They're tools of communication they're tools of creativity, and they can be shaped by their user."
– Bill Gates, February 2004

Acknowledgements

- Thanks to many sources for slide material

  © 1990 Morgan Kaufmann Publishers, © 2001-present Elsevier
  Computer Architecture: A Quantitative Approach by J. Hennessy & D. Patterson

  © 1994 Morgan Kaufmann Publishers, © 2001-present Elsevier
  Computer Organization and Design by D. Patterson & J. Hennessy

  © 2002 K. Asinovic & Arvind, MIT

  © 2002 J. Kubiatowicz, University of California at Berkeley

  © 2006, © 2010 No Starch Press for Inside the Machine by J. Stokes

  © 2007 W.-M. Hwu & D. Kirk, University of Illinois & NVIDIA

  © 2007-2010 J. Owens, University of California at Davis

  © 2010 CRC Press for Introduction to Concurrency in Programming Languages by M. Sottile, T. Mattson, and C. Rasmussen

  © 2017, IBM POWER9 Processor Architecture by Sadasivam et al., IBM

  © 2016, © 2019 POWER9 Processor User's Manual, IBM

  © The OpenPOWER Foundation

Technology

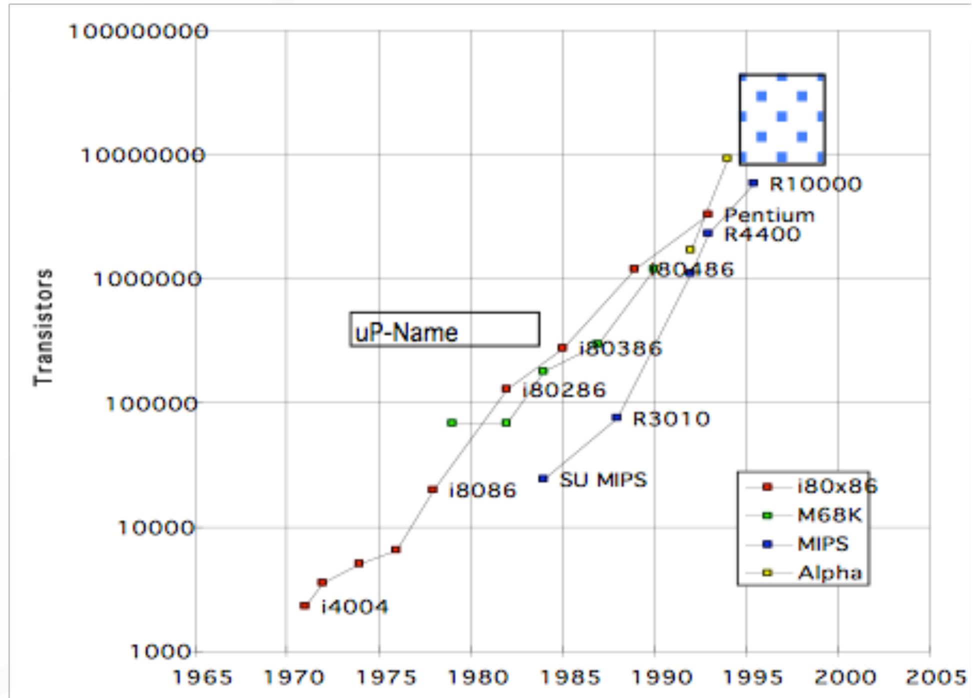
Workstations, PCs, etc. have been riding this wave since.

- In the mid-1980s, the single-chip processor (32-bit) and the single-board computer emerged.

DRAM Chip Capacity

1980	64 kB
1983	256 kB
1986	1 MB
1989	4 MB
1992	16 MB
1996	64 MB
1999	256 MB
2002	1 GB
2005	4 GB

Microprocessor Logic Density



Technology Rates of …

- Processor
  - Logic Capacity: ~30% per year
  - Clock Rate: ~20% per year (2003-2011);
    ~10% per year (2011-2015); ~3% per year (2015-now) … why?
- Memory
  - DRAM Capacity: ~60% per year (4x every 3 years)
  - Memory Speed: ~10% per year
  - Cost Per Bit: Improves ~25% per year
- Disk
  - Capacity: ~60% per year
  - Total Use of Data: ~100% per 9 months!
- Network
  - Bandwidth: 100%+ per year

Execution Time

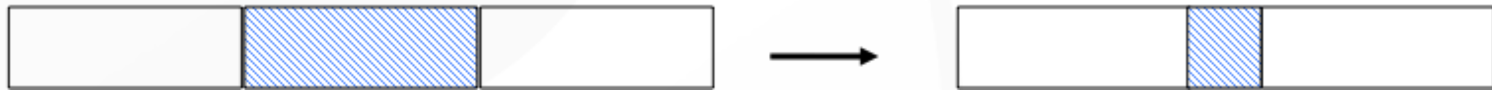
- Time = Clock Speed * CPI * Instruction Count
  = seconds/cycle * cycles/instr * instrs/program
  = seconds/program

- *Execution time is the only reliable measure of computer performance.*

What is Amdahl's Law?

- Speedup due to enhancement *E*

$$\text{Speedup}(E) = \frac{\text{Execution Time without } E}{\text{Execution Time with } E} = \frac{\text{Performance with } E}{\text{Performance without } E}$$



- Suppose enhancement *E* accelerates a fraction *F* of the computation by a factor of *S*

$$\text{Execution time (with } E) = ((1 - F) + F/S) \cdot \text{Execution time (without } E)$$

$$\text{Speedup (with } E) = \frac{1}{(1 - F) + F/S}$$

Just How Fast is "Fast"?

- New CPU comes out that is a whopping 10x faster
- Assume an I/O-bound server where 60% of time waits for I/O

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$= \frac{1}{(1 - 0.4) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

- Don't get "wow"-ed by 10x faster. It's only 1.56x faster in this case.

If we build a new architecture,
how do we tell how good it is?

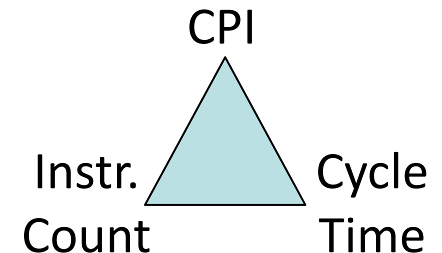
Basis of Architectural Evaluation

Approach	Pros	Cons
Actual Target Workload	• Representative	• Very specific • Non-portable • Difficult to run or measure • Hard to identify cause
Full Application Benchmarks	• Portable • Widely used • Improvements useful in reality	• Less representative
Small "kernel" benchmarks	• Easy to run • Early in design cycle	• Easy to "fool"
Microbenchmarks	• Identify peak capability & potential bottlenecks	• "Peak" may be a long way from application performance
Dwarfs* → Motifs (e.g., OpenDwarfs)	?	?

* See "The Landscape of Parallel Computing Research: A View from Berkeley," December 2006.

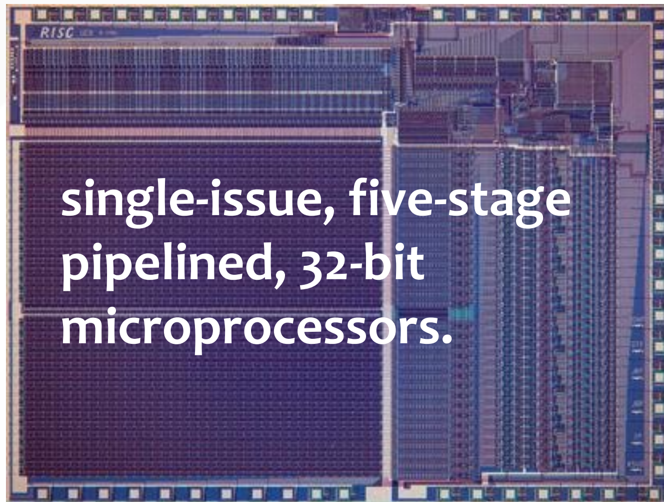
Evaluating Instructions

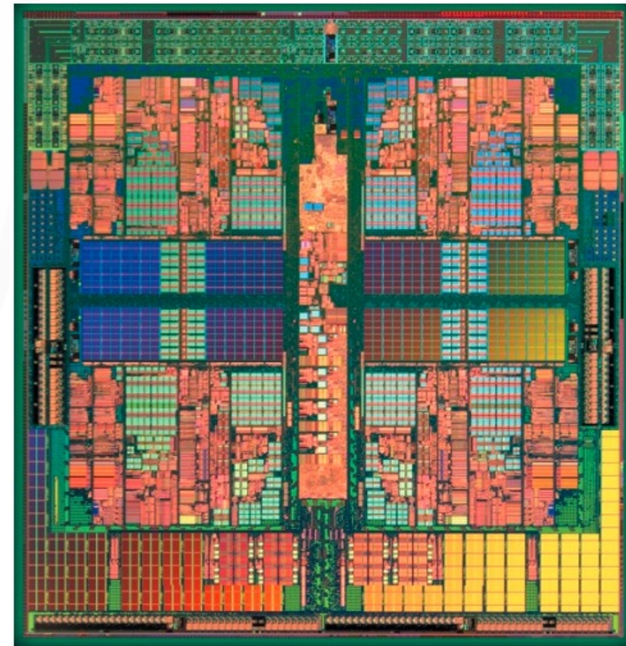
- Design-Time Metrics → Productivity
  - Can it be implemented? In how much time? At what cost?
  - Can it be programmed? Ease of compilation?
- Static Metrics → Efficiency
  - How many bytes does the program occupy in memory?
- Dynamic Metrics → Efficiency/Performance
  - How many instructions are executed?
  - How many bytes does the processor fetch to execute the program?
  - How many clocks are required per instruction?
  - How "lean" a clock is practical?
- Best Metric for Computer Architecture?
  - Time to execute the program!



RISC "Revolution"

- Questions
  - How did this "revolution" come about?
  - What happened to the revolution?
    - x86 instruction set architecture (ISA)





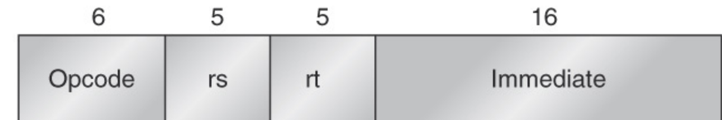
RISC: MIPS Instruction Set

- 32-bit fixed format inst (3 formats)
- 32 32-bit GPR (R0 contains zero); 32 FP registers (and HI LO)
  - partitioned by software convention
- 3-address, reg-reg arithmetic instr.
- Single address mode for load/store: base+displacement
  - no indirection, scaled
- 16-bit immediate
- Simple branch conditions
  - compare against zero or two registers for =,≠
  - no integer condition codes
- Delayed branch
  - execute instruction after a branch (or jump) even if the branch is taken
  - compiler can fill branch delay slot ~50% of the time

RISC Philosophy

- Instructions all same size
- Small number of opcodes (small opcode space)
- Opcode in same place for every instruction
- Simple memory addressing
- Instructions that manipulate data don't manipulate memory, and vice versa
- Minimize memory references by providing ample registers
- *Implications to pipelining?*

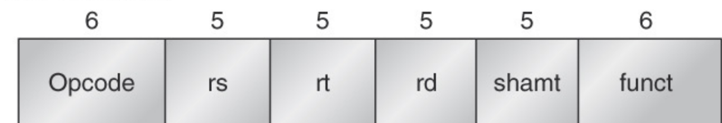
I-type instruction



Encodes: Loads and stores of bytes, half words, words, double words. All immediates (rt ← rs op immediate)

Conditional branch instructions (rs is register, rd unused)
Jump register, jump and link register
(rd = 0, rs = destination, immediate = 0)

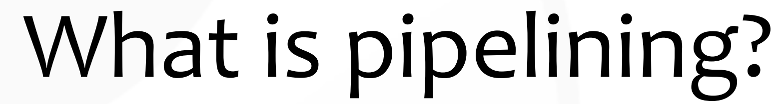
R-type instruction



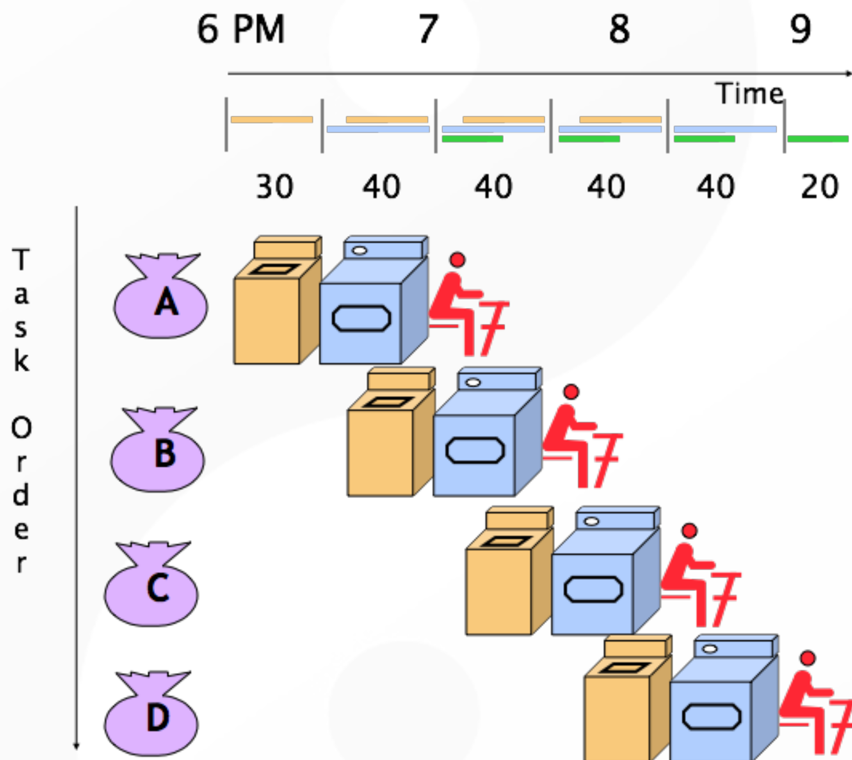
Register-register ALU operations: rd ← rs funct rt
Function encodes the data path operation: Add, Sub, . . .
Read/write special registers and moves

J-type instruction

Jump and jump and link
Trap and return from exception

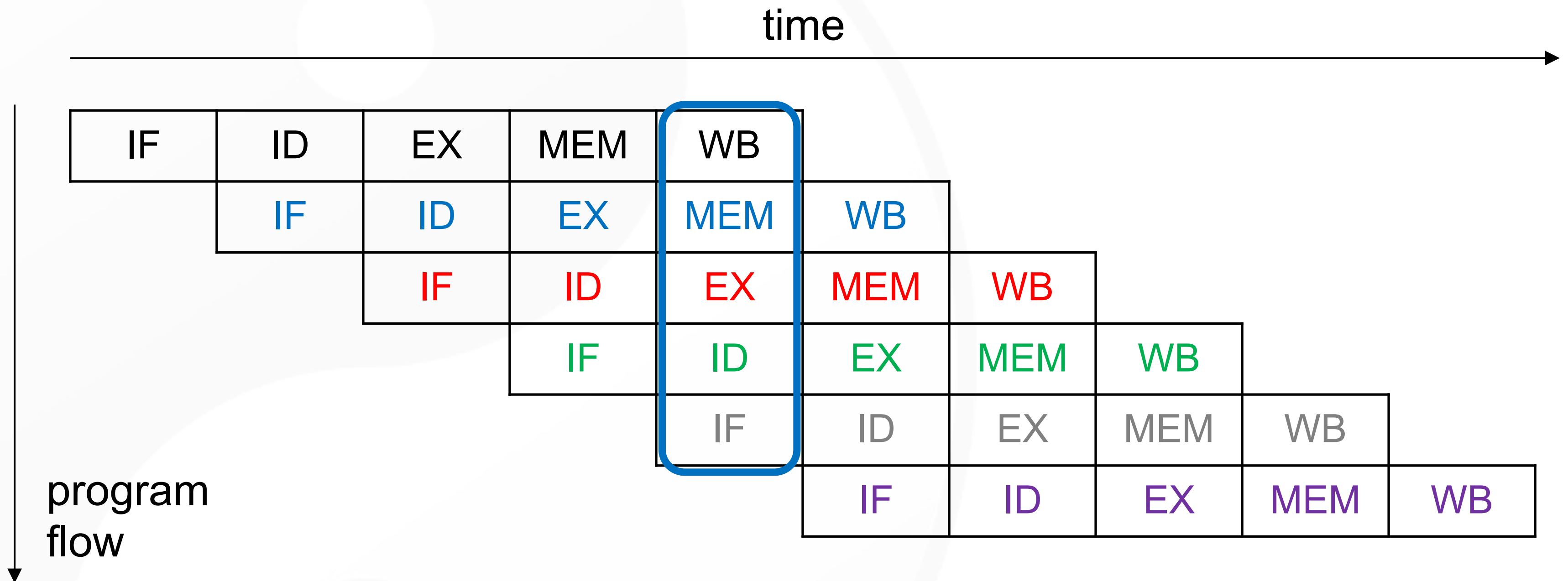

What is pipelining?

Pipelining Review

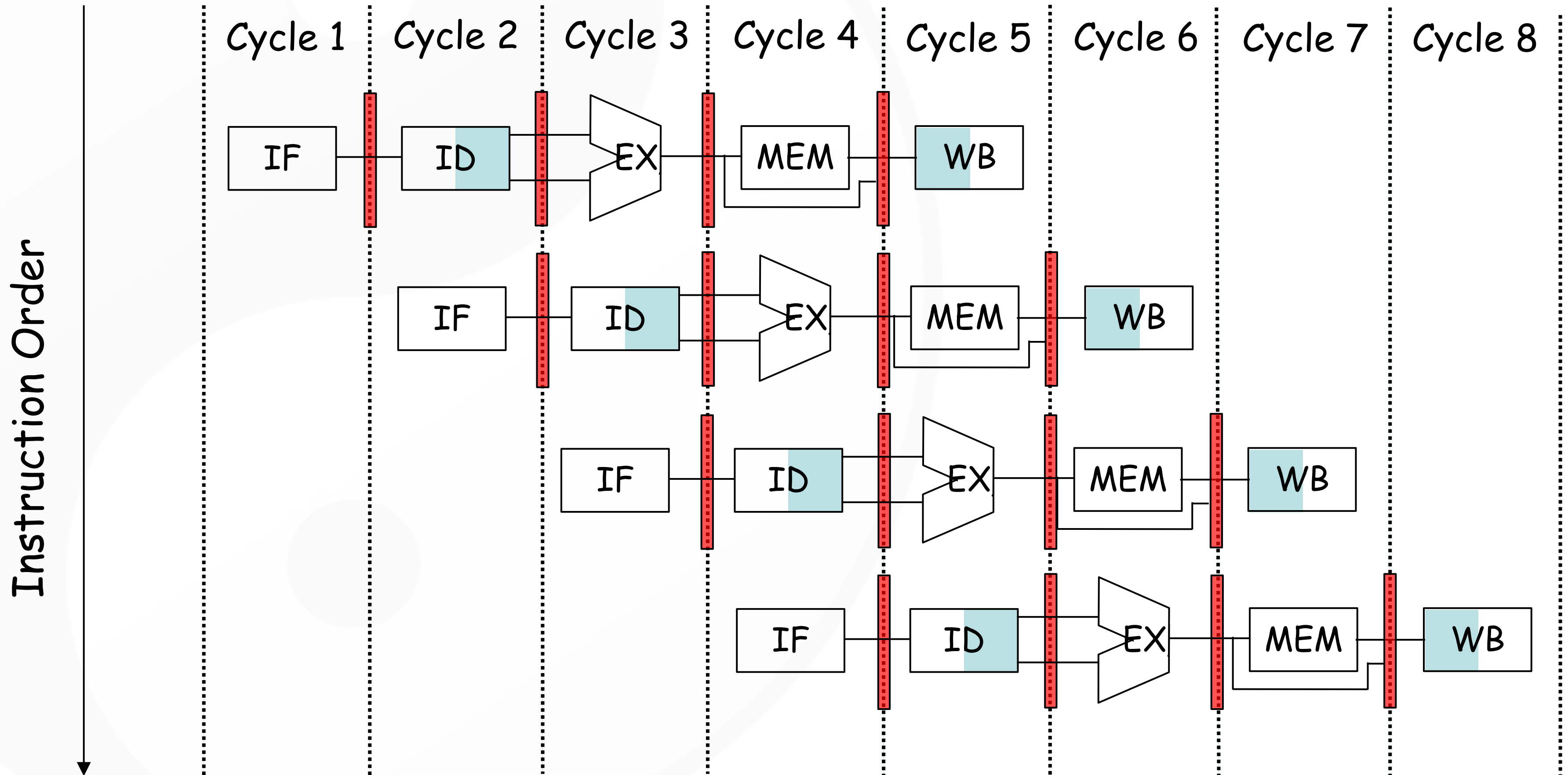


- Pipelining *does NOT help latency* of a single task, it *helps throughput* of entire workload
- Pipeline rate limited by *slowest* pipeline stage
- Multiple tasks run simultaneously using different resources
- Potential speedup =
  number of pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup
- Stall for dependencies

Conventional Pipelined Execution



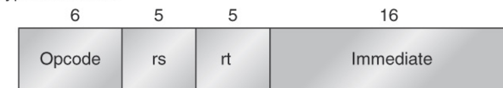
Pipelined Architecture



Why MIPS is “Awesome”

- All MIPS instructions same length
- Source registers located in same place for every instruction
- Overlap register fetch & instruction decode
- Simple memory operations
- MIPS: execute calculates memory address, memory load/store in next stage
- X86: can operate on result of load: execute calculates memory address, memory load/store in next stage, THEN ALU stage afterwards
- All instructions aligned in memory — one access for each instruction

I-type instruction



Encodes: Loads and stores of bytes, half words, words, double words. All immediates (rt ← rs op immediate)

Conditional branch instructions (rs is register, rd unused)
Jump register, jump and link register
(rd = 0, rs = destination, immediate = 0)

R-type instruction

Register-register ALU operations: rd ← rs funct rt
Function encodes the data path operation: Add, Sub, . . .
Read/write special registers and moves

J-type instruction

Jump and jump and link
Trap and return from exception

Limits to Pipelining

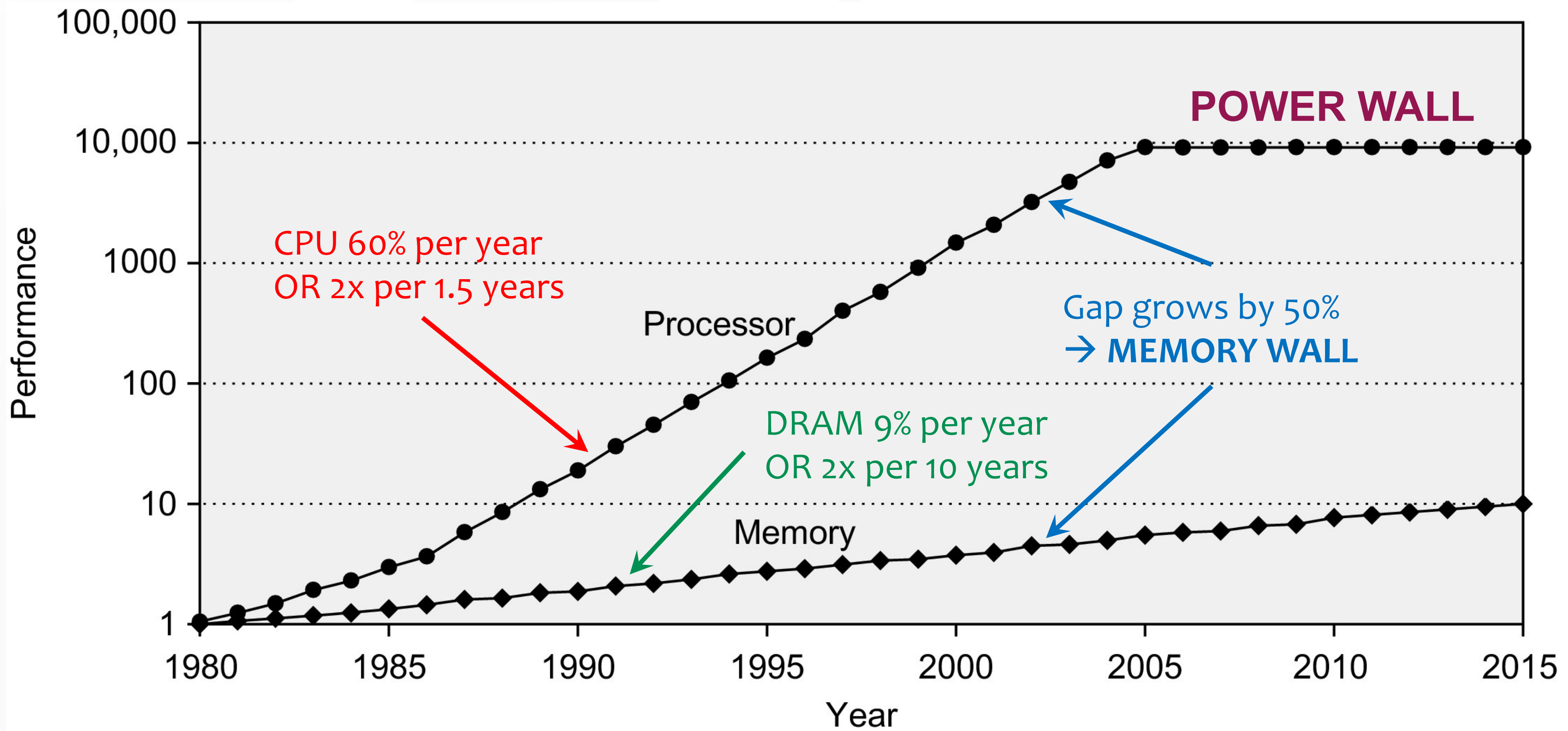
- Hazards prevent next instruction from executing during its designated clock cycle
  - ***Structural hazards:*** Two instructions attempt to use the same hardware to do two different things at once
  - ***Data hazards:*** Instruction depends on result of prior instruction still in the pipeline
  - ***Control hazards:*** This is caused by the delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).

Focus on the Common Case

- Common sense guides computer design
- In making a design trade-off, favor the frequent case over the infrequent case, e.g.,
  - Instruction fetch and decode unit used more frequently than multiplier, so optimize it *first*
  - If database server has 50 disks/processor, storage dependability dominates system dependability, so optimize it first
- Frequent case is often simpler and can be done faster than the infrequent case, e.g.,
  - Overflow is rare when adding two numbers, so improve performance by optimizing common case of no overflow
- What is the frequent case and how much performance improved by making case faster?
  - Amdahl's Law

What is the memory hierarchy?
(Why does a memory hierarchy exist?)

Why Care about Memory?



Memory Levels

Capacity
Access-Time Cost

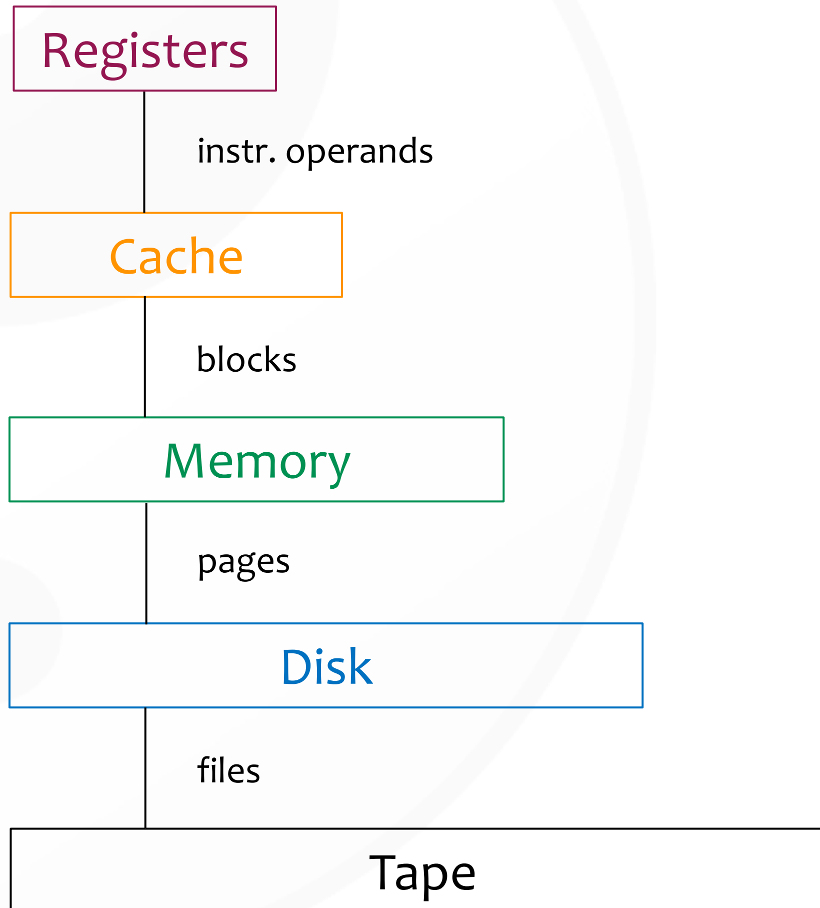
CPU Registers
100s bytes (B)
< 2 ns

Cache
kB SRAM
2-100 ns
$0.01-$0.001/bit

Main Memory
GB DRAM
100 ns – 1000 ns
$0.01-$0.001/bit

Disk
TB
10 ms – 40 ms
$0.001-$0.0001/bit

Tape
"infinite"
sec – min
$0.000001/bit



Staging
Transfer Unit

program/compiler
1-8 bytes

cache controller
8-128 bytes

OS
512-4k+ bytes

user/operator
MB – GB

faster

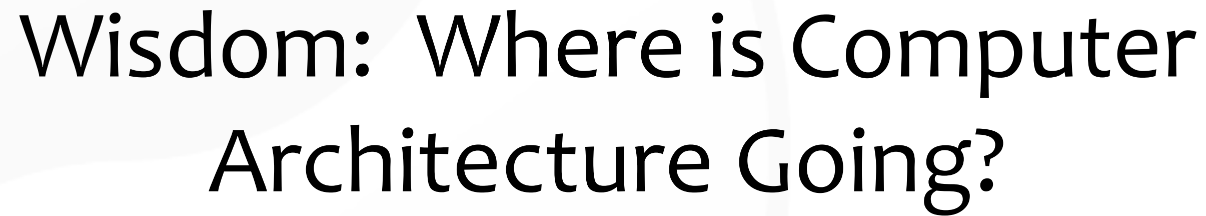
larger

Memory Hierarchy

- The Principle of Locality
  - Program access a relatively small portion of the address space at any instant of time
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses
  - Compulsory Misses: cannot avoid
    - Example: cold start misses
  - Conflict Misses: increase cache size and/or associativity
  - Capacity Misses: increase cache size

Cost of Fabrication

- Rock’s Law: Cost of fabs double every 4 years
- $3B for fab *in 2008-2009*
  - Rise of fabless design houses (AMD now fabless)
  - Rise of for-hire fabs (TSMC, etc.)
  - $3B fab means $6B in revenue is required


Wisdom: Where is Computer
Architecture Going?

Looking to the Future

The Landscape of Parallel Computing Research: A View from Berkeley

Krste Asanovic
Ras Bodik
Bryan Christopher Catanzaro
Joseph James Gebis
Parry Husbands
Kurt Keutzer
David A. Patterson
William Lester Plishker
John Shalf
Samuel Webb Williams
Katherine A. Yelick

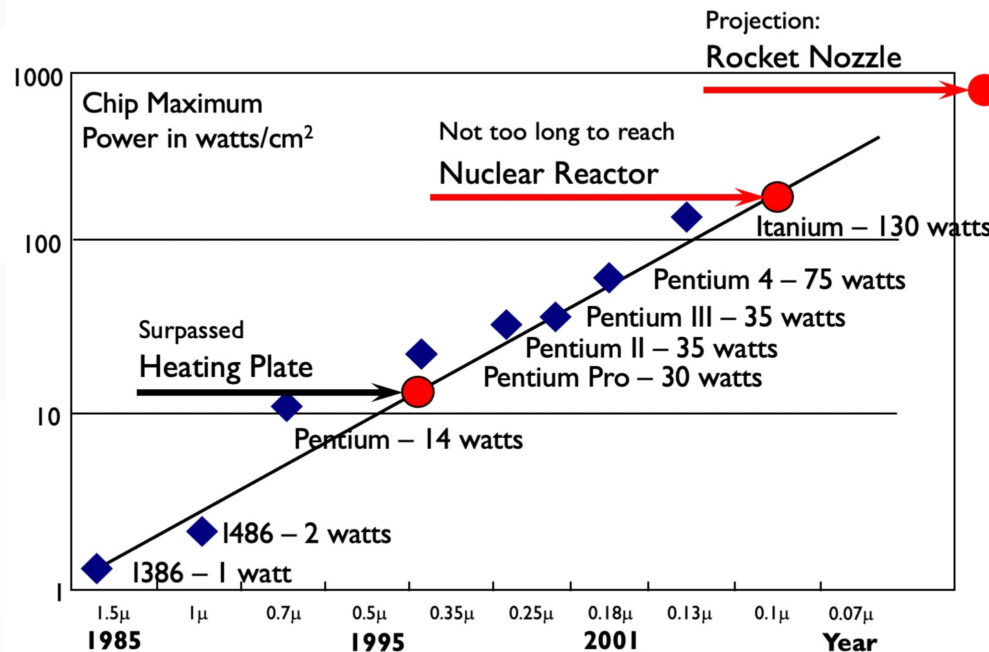
Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-183
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

December 18, 2006

Power

- Old Conventional Wisdom (CW)
  - Power is free, but transistors are expensive.
- New CW is the "Power Wall"
  - Power is expensive, but transistors are free. We can put more transistors on a chip than we have power to turn on.



Source: Fred Pollack, Intel. New Microprocessor Challenges in the Coming Generations of CMOS Technologies, MICRO32 and Transmeta

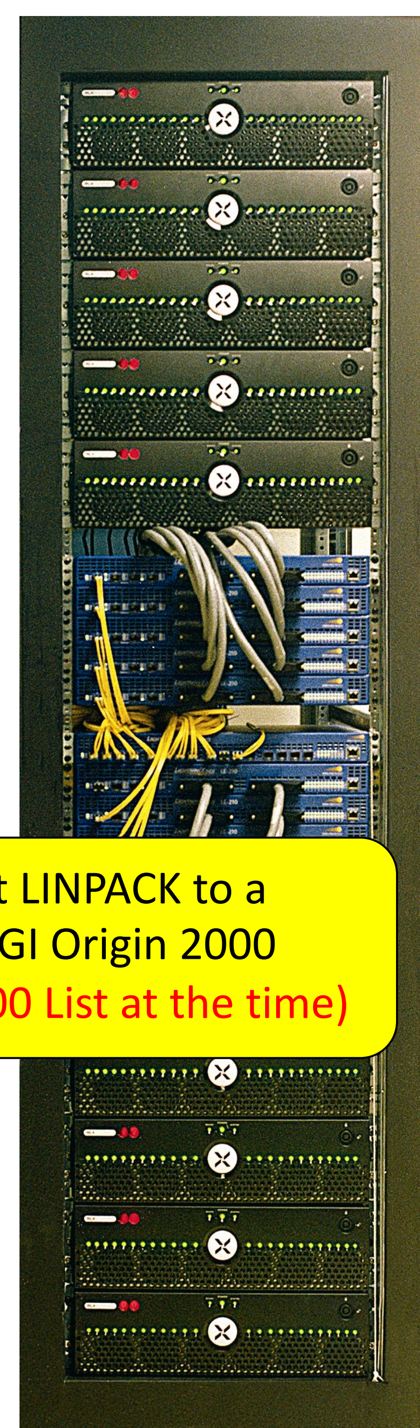
Green Destiny Supercomputer

(circa December 2001 – February 2002)

- A 240-Node Cluster in Five Sq. Ft.
- Each Node
  - 1-GHz Transmeta TM5800 CPU w/ High-Performance Code-Morphing Software running Linux 2.4.x
  - 640-MB RAM, 20-GB hard disk, 100-Mb/s Ethernet
- Total
  - 240 Gflops peak (LINPACK: 101 Gflops in March 2002.)
  - 150 GB of RAM (expandable to 276 GB)
  - 4.8 TB of storage (expandable to 38.4 TB)
  - ***Power Consumption: Only 3.2 kW (diskless)***
- Reliability & Availability
  - ***No unscheduled downtime in 24-month lifetime.***
    - **Environment: A dusty 85-90° F warehouse!**

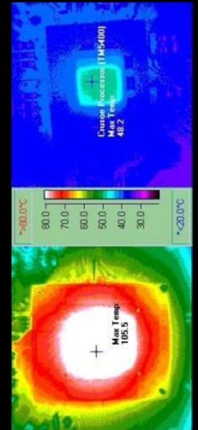
Equivalent LINPACK to a 256-CPU SGI Origin 2000
(On the TOP500 List at the time)

Featured in The New York Times, BBC News, and CNN.
Now in the Computer History Museum.



Green Destiny: Low-Power Supercomputer

Only Difference? The Processors



Green Destiny "Imitation": Traditional Supercomputer

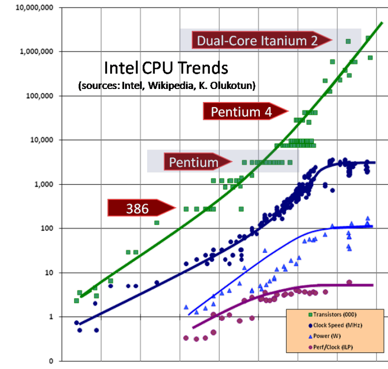
Compute vs. Memory

- Old CW
  - Multiply is slow, but load and store from/to memory is fast.
- New CW
  - The Memory Wall.
  - Load and store is slow, but multiply is fast.
    - Modern microprocessors can take 100 clock cycles to access dynamic random-access memory (DRAM), but even floating-point multiplies may take only four clock cycles.

Evolution of Computation

- 20 years ago: computation expensive, wires free
  - To first order: ignore wire delay
- Light moves 1 foot/ns in vacuum
  - Wires are also getting thinner
  - *Wire delay now significant even on chip!*
- Moore's Law implies
  - Computation gets cheaper
  - Speed of light doesn't change
  - Compute don't communicate!

Intel Finally "Gets It" …



Intel Developer Forum, September 2004

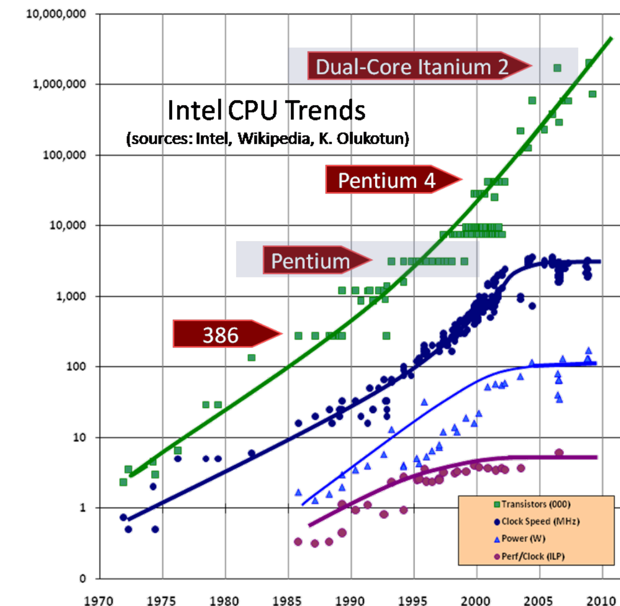
- "What Intel announced at this IDF was no less than a total rethinking of their approach to microprocessors."
- Performance improvement from Moore's Law performance scaling will come *not* from increases not in MHz ratings but in machine width.
  - Power wall
  - Threading
  - MIPS/watt instead of MIPS

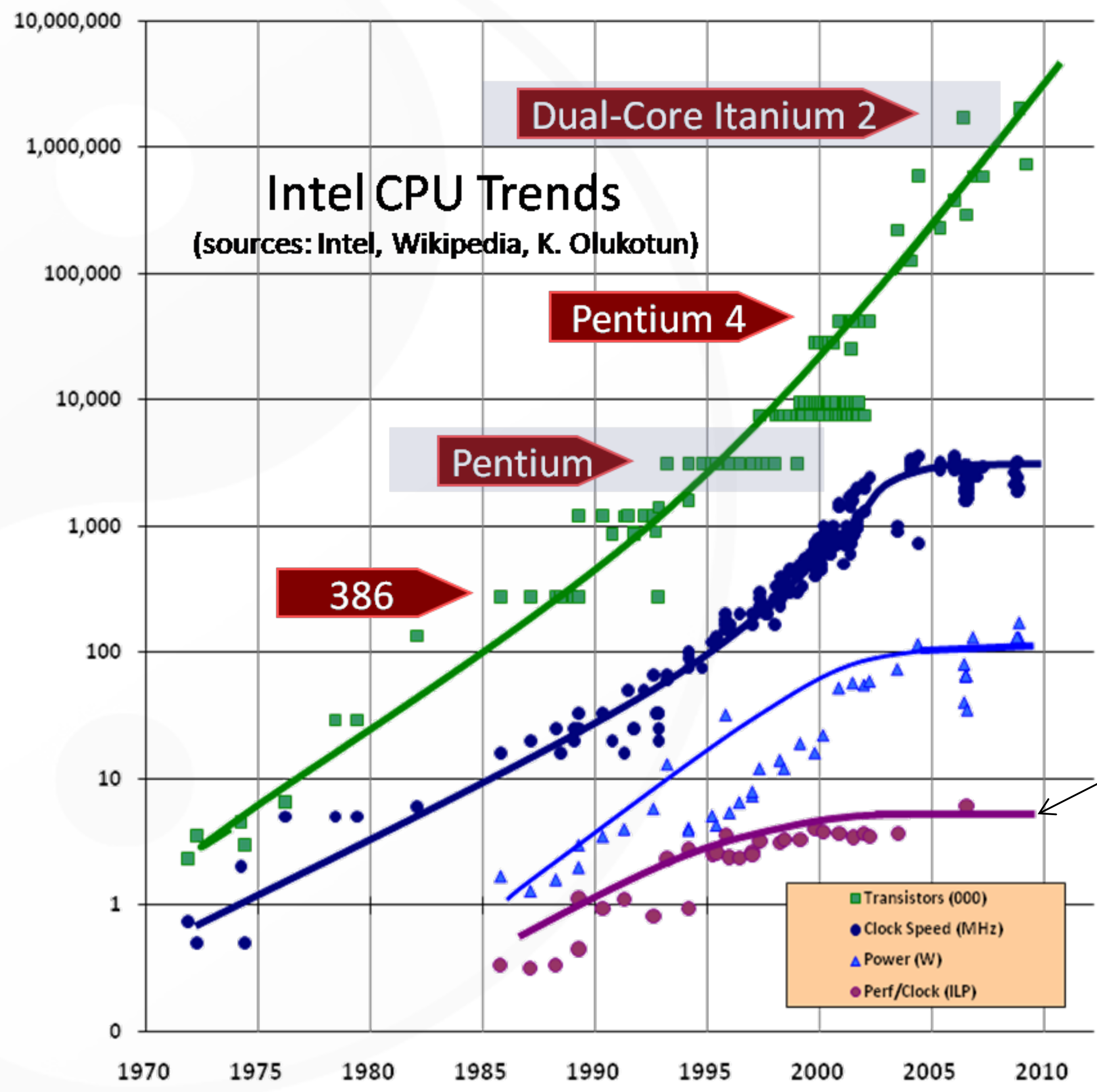
http://arstechnica.com/articles/paedia/cpu/intel-future.ars/1

- Datasets are growing in size and so are the network pipes that connect those datasets.
  - Intel claimed "doubling of digital data every 18 months"
  - More integration?

Instruction Level

- Old CW: We can reveal more instruction-level parallelism (ILP) via compilers and architecture innovation.
  - Examples from the past include branch prediction, out-of-order execution, speculation, and Very Long Instruction Word (VLIW) systems.
  - This CW will be approximately the first 5 weeks of the semester.
- New CW is the "ILP wall": There are diminishing returns on finding more ILP.





ILP

Uniprocessor

- Old CW

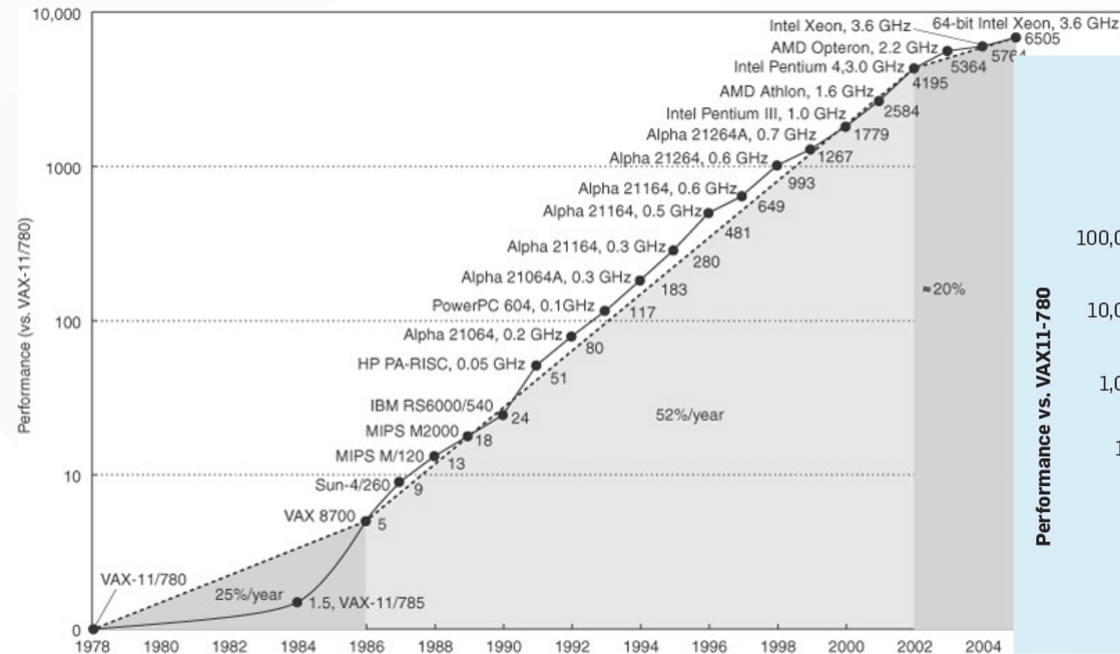
  Uniprocessor performance doubles every 18 months.

- New CW

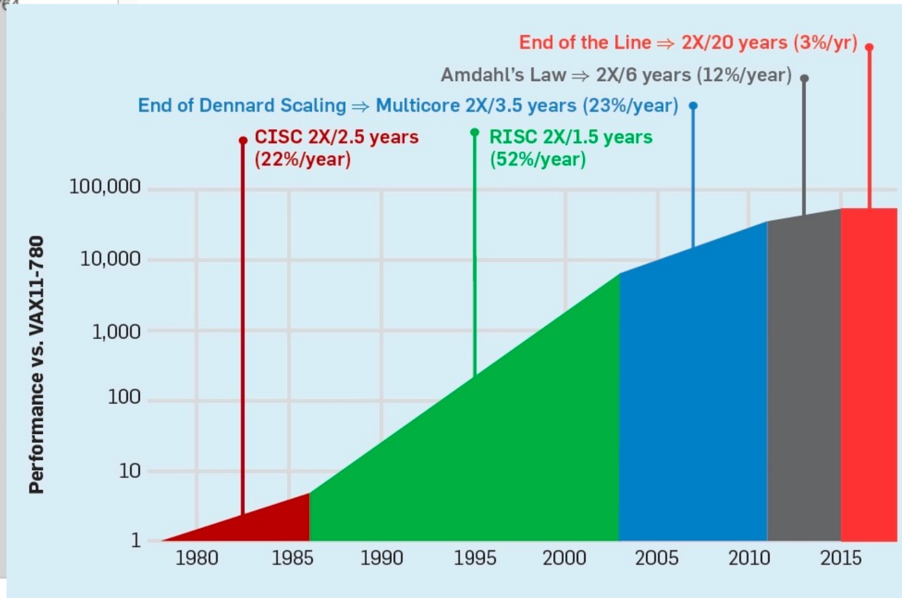
  Power Wall + Memory Wall + ILP Wall = Brick Wall.

In 2006, performance is a factor of three below the traditional doubling every 18 months that we enjoyed between 1986 and 2002.

The doubling of uniprocessor performance may now take 20 years.



© 2007 Elsevier, Inc. All rights reserved.



On the Evolution of Computer Architecture Wisdom

- Old CW: Don’t bother parallelizing your application, as you can just wait a little while and run it on a much faster sequential computer.

  New CW: It will be a very long wait for a faster sequential computer.
- Old CW: Increasing clock frequency is the primary method of improving processor performance.

  New CW: Increasing parallelism is the primary method of improving processor performance.
- Old CW: Less than linear scaling for a multiprocessor application is failure.

  New CW: Given the switch to parallel computing, any speedup via parallelism is a success.