Modern Computer Architecture (Processor Design)

Prof. Dan Connors
dconnors@colostate.edu

Computer Architecture

- Historic definition

  Computer Architecture =

    Instruction Set Architecture +

    Computer Organization

- Famous architects: Wright, Fuller, Herzog
- Famous computer architects: Smith, Patt, Hwu, Hennessey, Patterson

Instruction Set Architecture

- Important acronym: **ISA**
  - Instruction Set Architecture
- The low-level software interface to the machine
  - Assembly language of the machine
  - Must translate any programming language into this language
  - Examples: IA-32 (Intel instruction set), MIPS, SPARC, Alpha, PA-RISC, PowerPC, …
- Visible to programmer

Differences between ISA’s

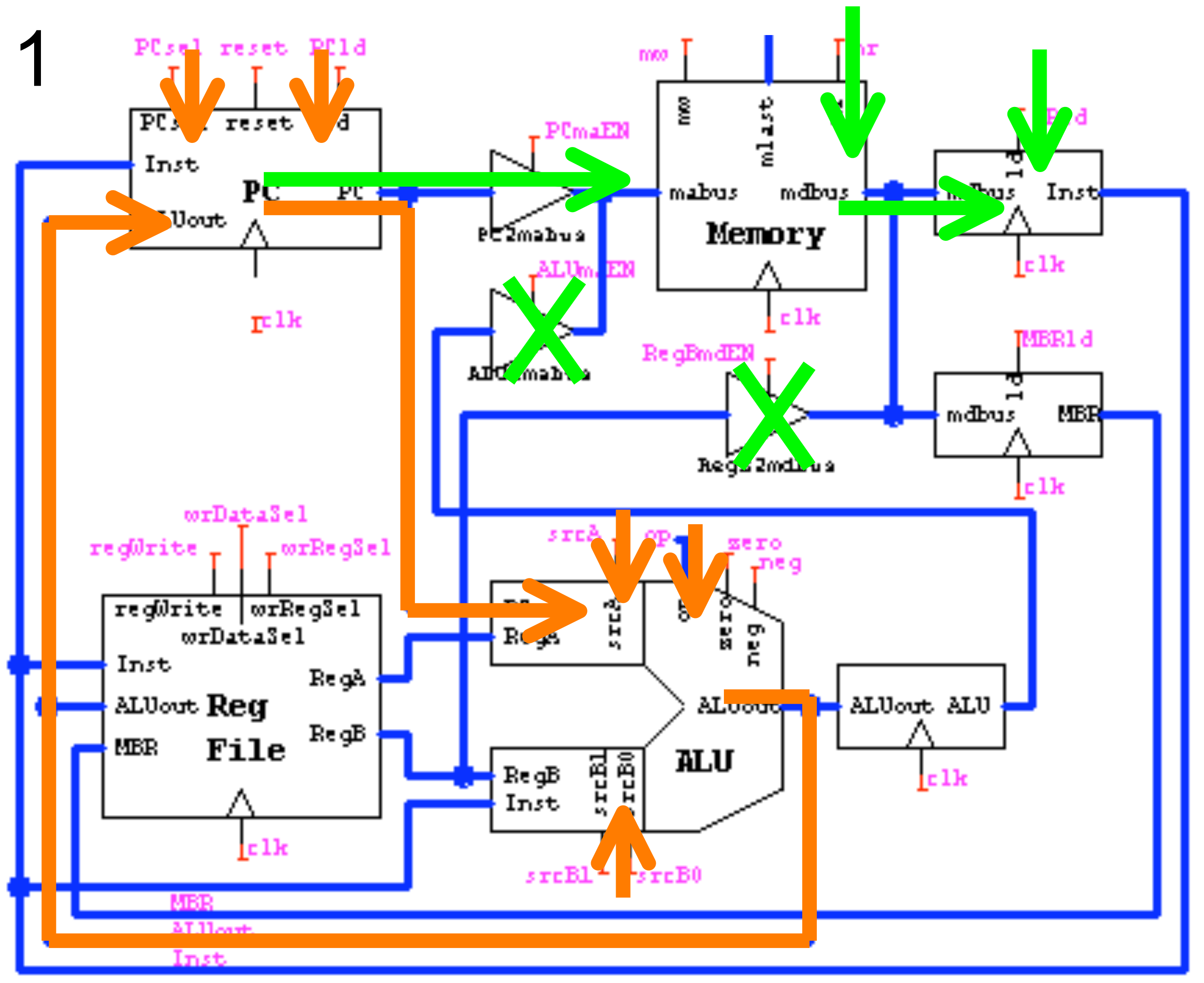
- Much more is similar between ISA’s than different. Compare MIPS & x86:
  - Instructions:
    - same basic types
    - different names and variable-length encodings
    - x86 branches use condition codes (like MIPS floating point)
    - x86 supports (register + memory) -> (register) format
  - Registers:
    - Register-based architecture
    - different number and names, x86 allows partial reads/writes
  - Memory:
    - Byte addressable, 32-bit address space
    - x86 has additional addressing modes and many instruction types can access memory

RISC vs. CISC

- MIPS was one of the first RISC architectures. It was started about 20 years ago by John Hennessy, one of the authors of our textbook.
- The architecture is similar to that of other RISC architectures, including Sun's SPARC, IBM and Motorola's PowerPC, and ARM-based processors.
- Older processors used complex instruction sets, or CISC architectures.
  - Many powerful instructions were supported, making the assembly language programmer's job much easier.
  - But this meant that the processor was more complex, which made the hardware designer's life harder.
- Many new processors use reduced instruction sets, or RISC architectures.
  - Only relatively simple instructions are available. But with high-level languages and compilers, the impact on programmers is minimal.
  - On the other hand, the hardware is much easier to design, optimize, and teach in classes.
- Even most current CISC processors, such as Intel 8086-based chips, are now implemented using a lot of RISC techniques.

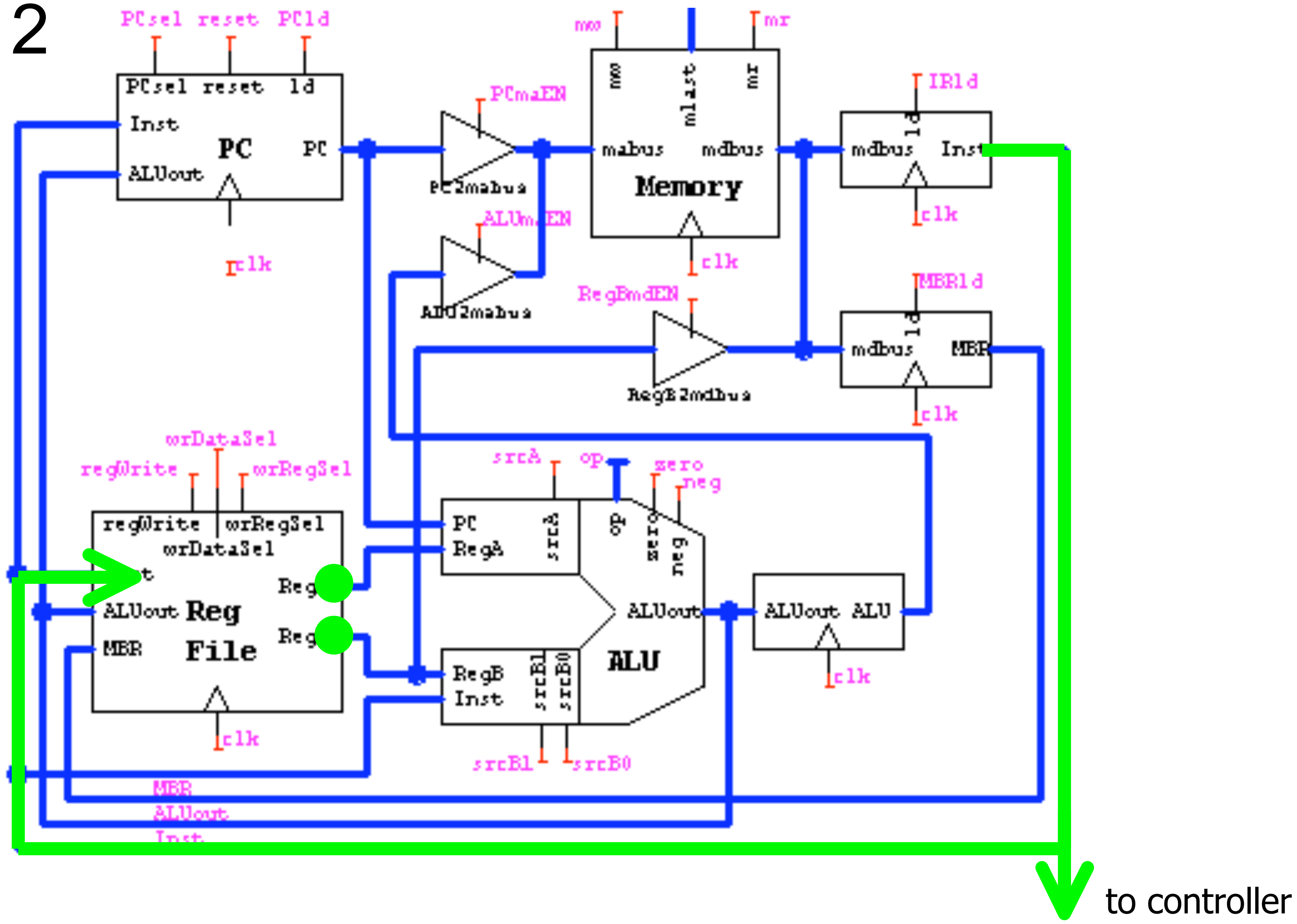
Tracing an Instruction's Execution

- Step 1



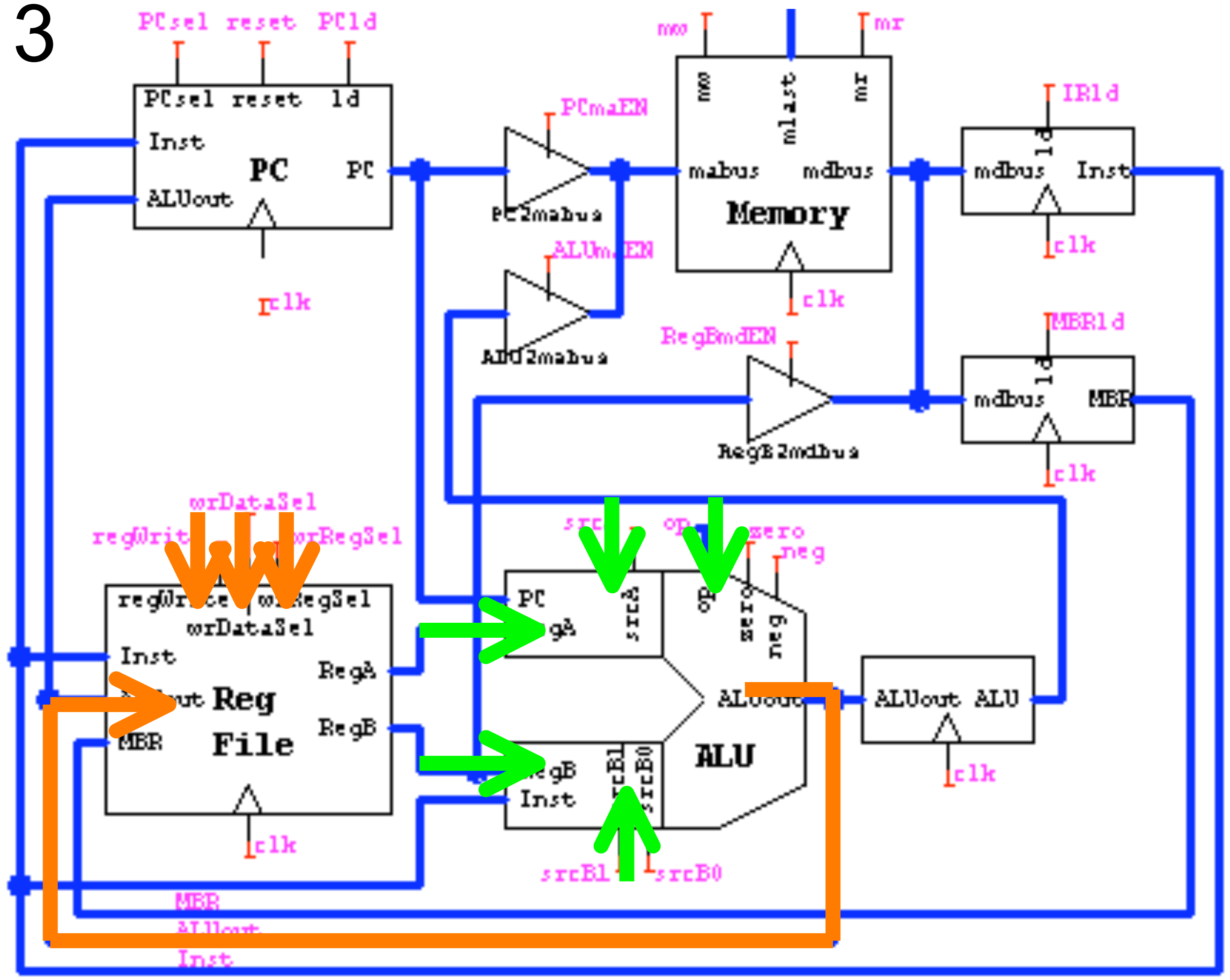
Tracing an Instruction's Execution

- Step 2

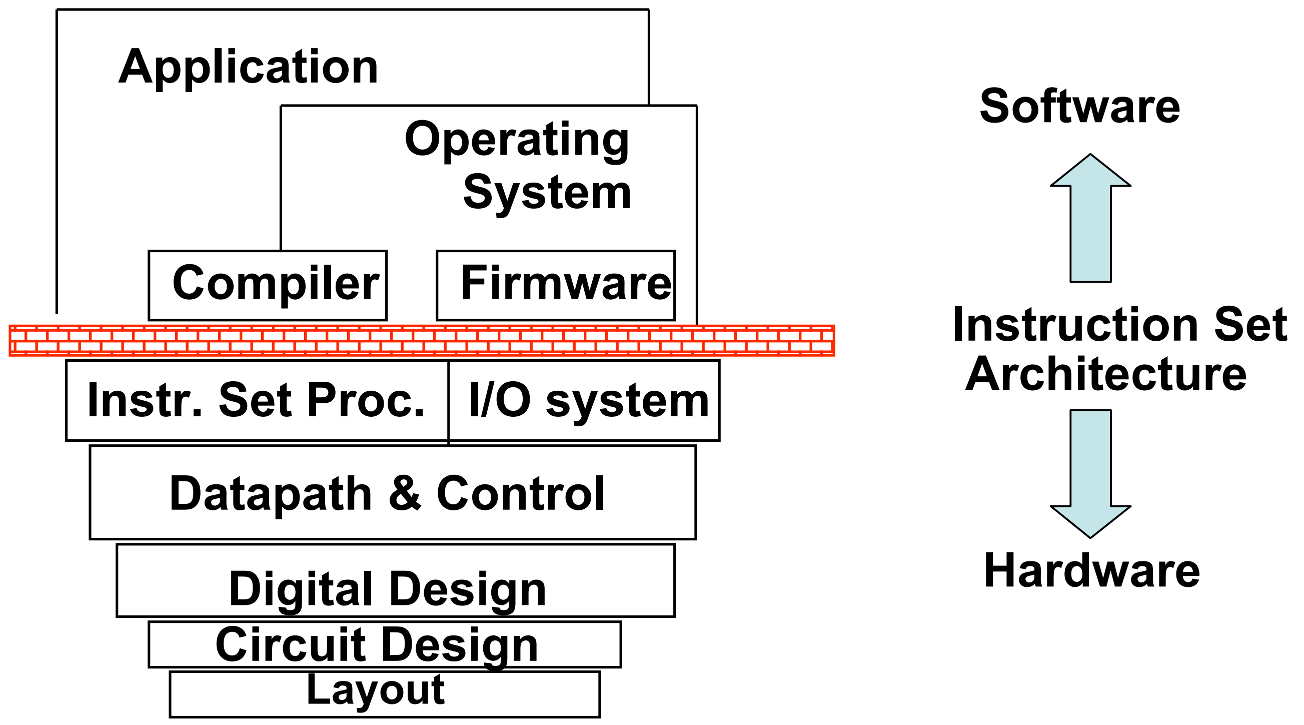


Tracing an Instruction's Execution

- Step 3



Instruction Set Architecture



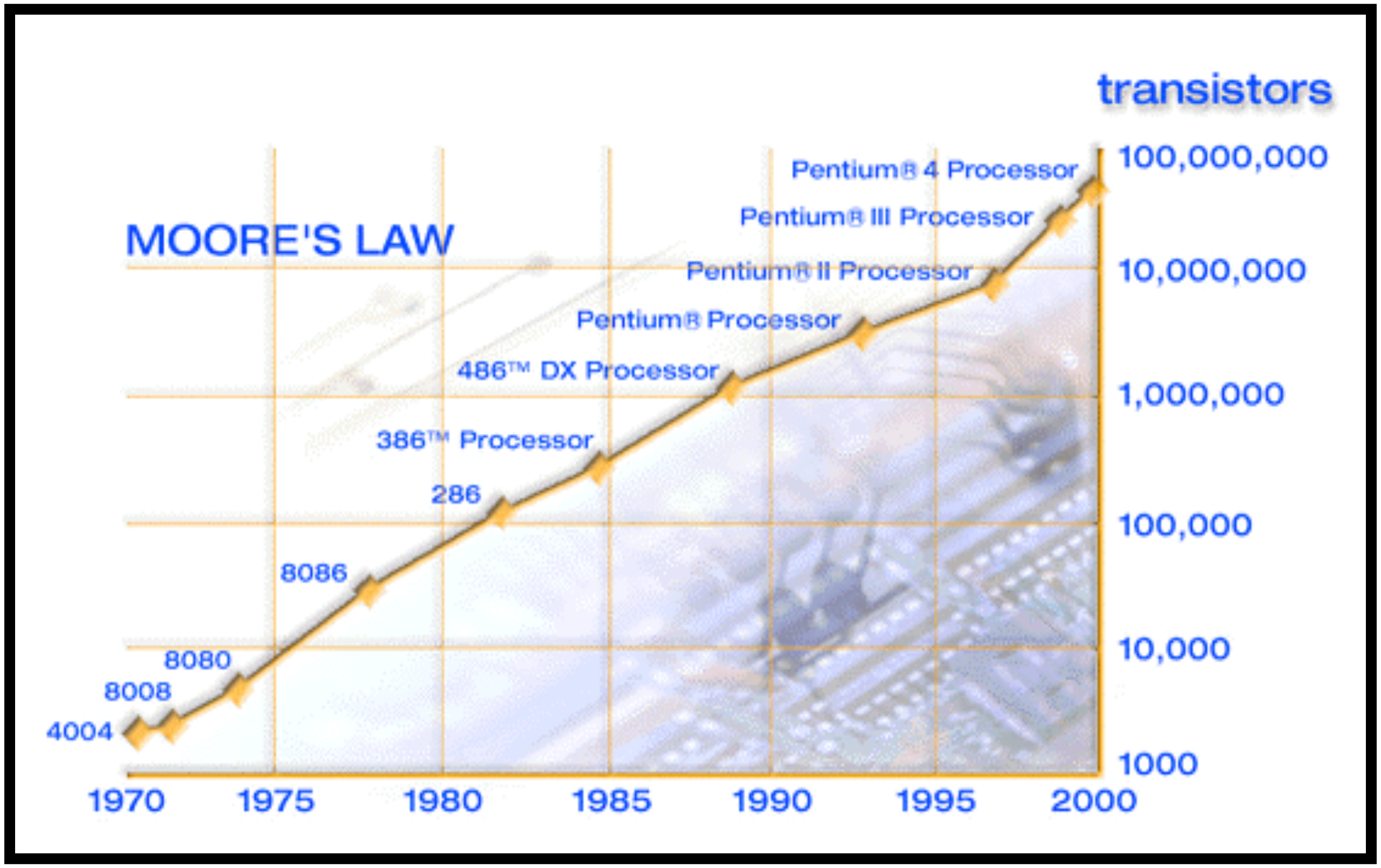
Computer Organization

- Computer organization is the *implementation* of the machine, consisting of two components:
  - High-level organization
    - Memory system widths, bus structure, CPU internal organization, …
  - Hardware
    - Precise logic and circuit design, mechanical packaging, …

- Many implementations are possible for an ISA !!!
  - Intel i386, i486, Pentium, Pentium II, Core2Duo, QuadCore…
  - Performance, complexity, cost differences….

- Invisible to the programmer (mostly)!

Moore’s Law: 2x transistors every 18 months



Technology => dramatic change

- **Processor**
  - logic capacity: about 30% increase per year
  - clock rate: about 20% increase per year

Higher logic density gave room for instruction pipeline & cache

- **Memory**

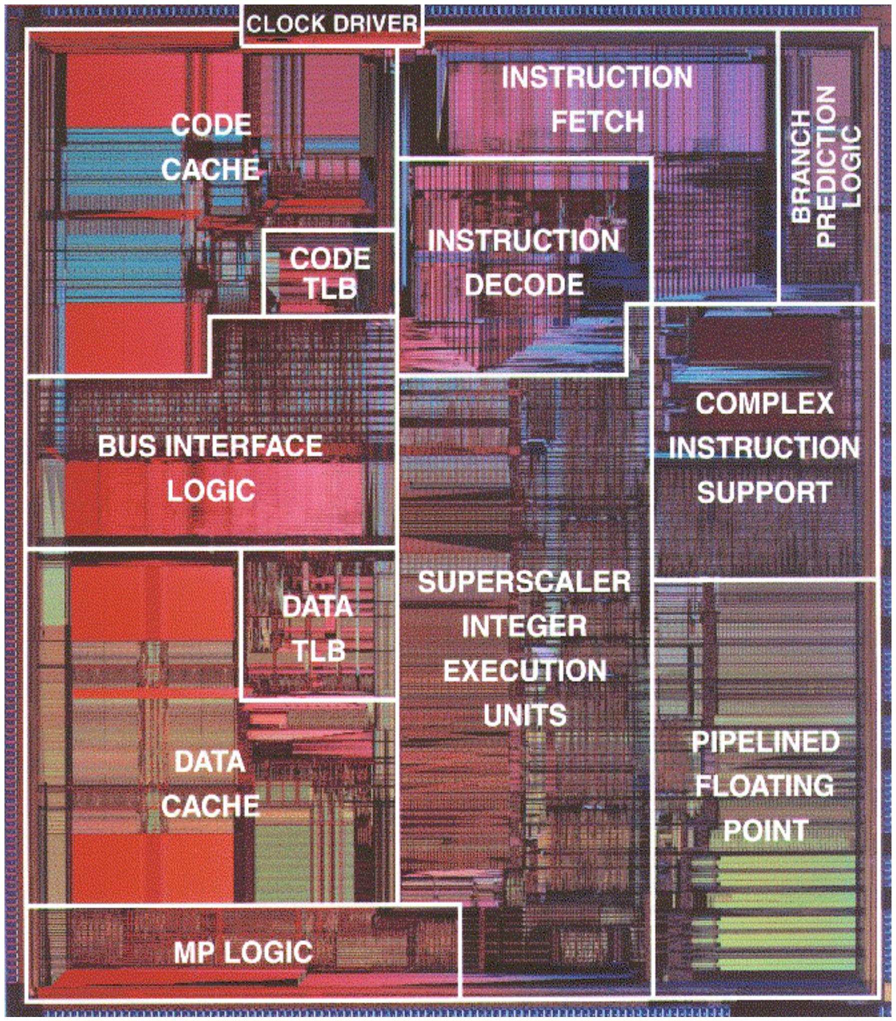
DRAM capacity:	about 60% increase per year (4x every 3 years)
Memory speed:	about 10% increase per year
Cost per bit:	about 25% improvement per year

Performance optimization no longer implies smaller programs

- **Disk**
  - Capacity: about 60% increase per year

Computers became lighter and more power efficient

Pentium Die Photo

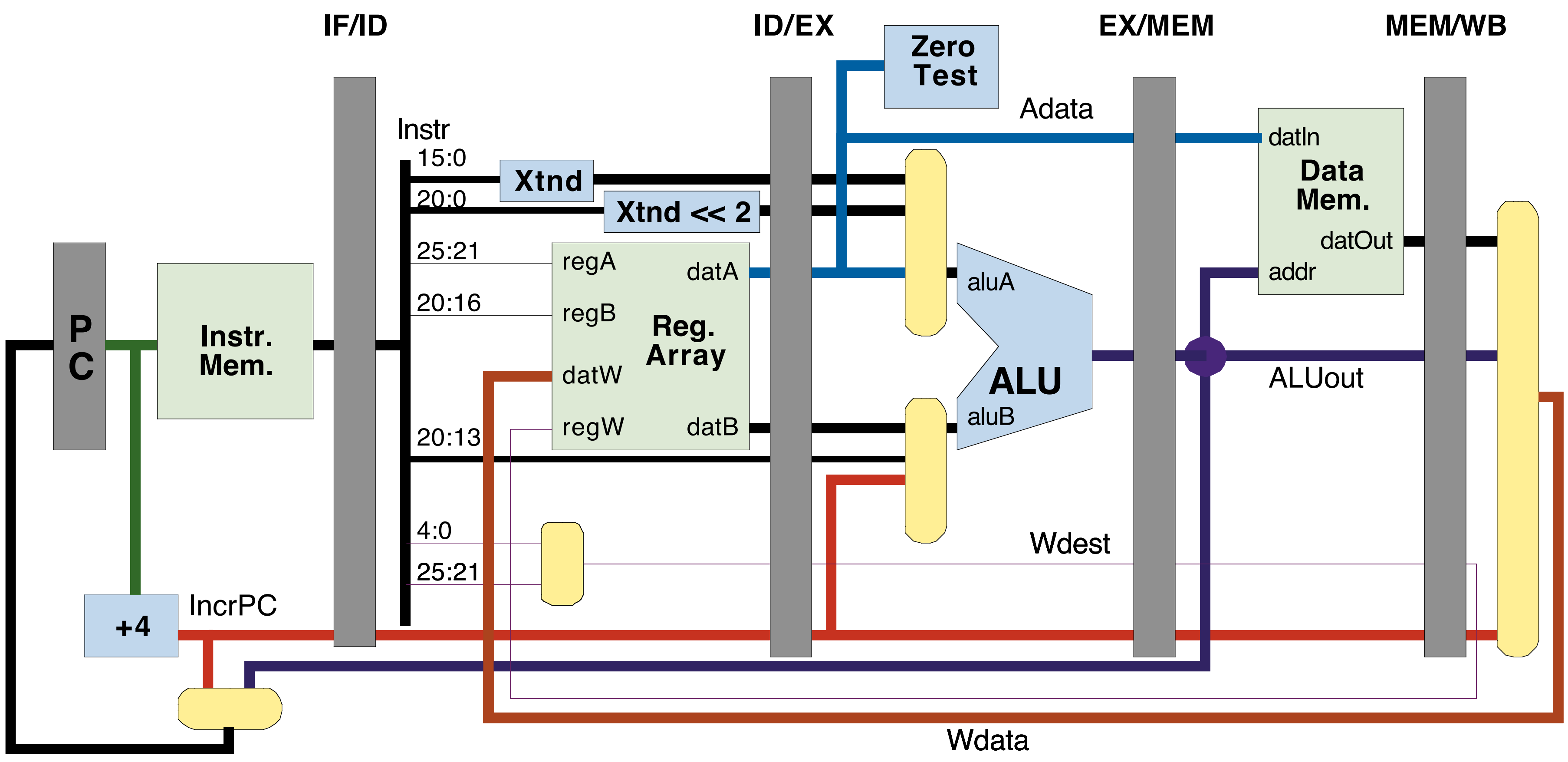


- 3,100,000 transistors
- 296 mm^2
- 60 MHz
- Introduced in 1993
  - 1st superscalar implementation of IA32

Four Organization Concepts/Trends

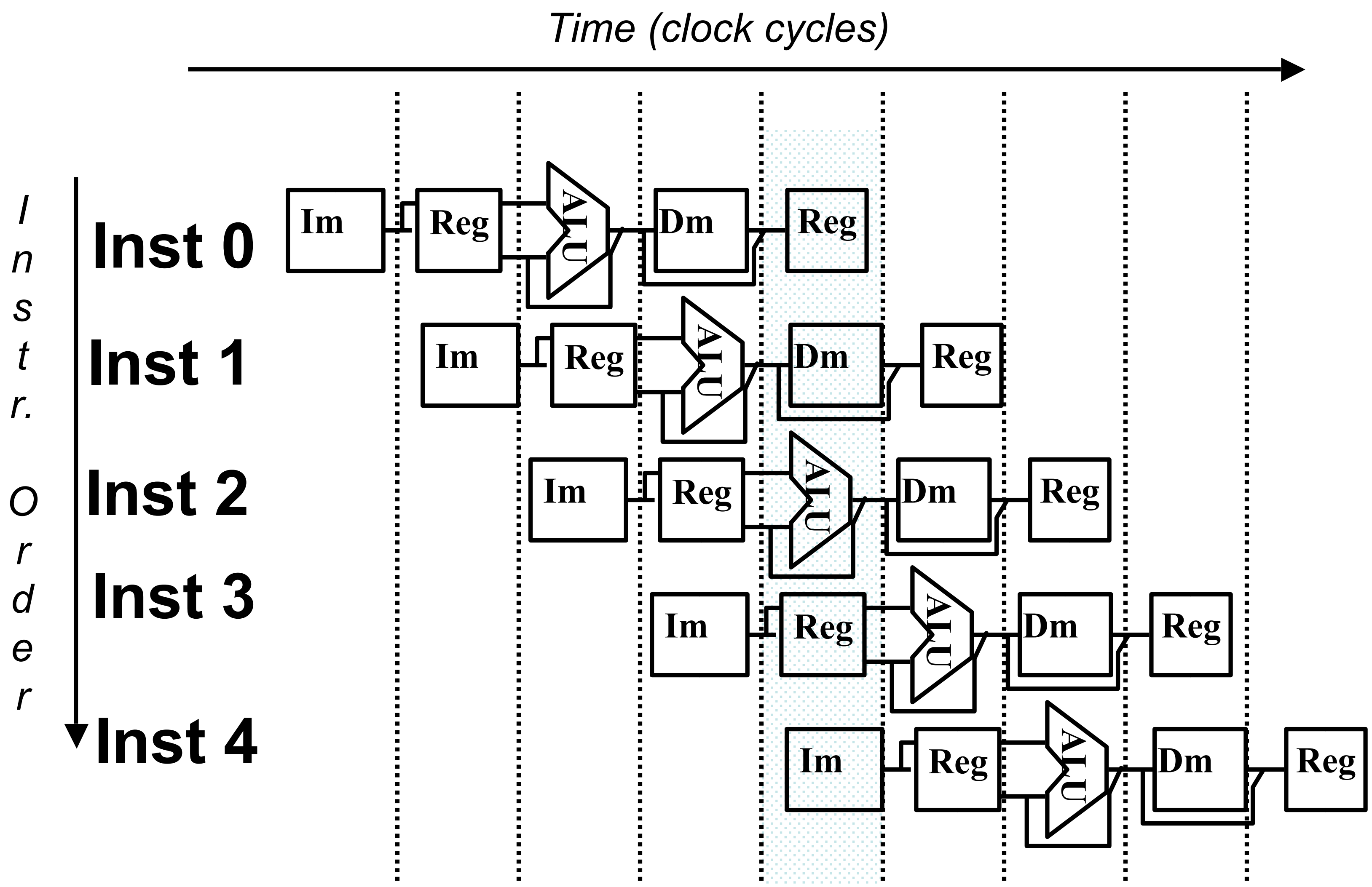
- Pipelining
  - Frequency
- Cache memory
  - To keep processor running must overcome memory latency
- Superscalar execution
  - Out of order execution
- Multi-core

Pipelined Datapath

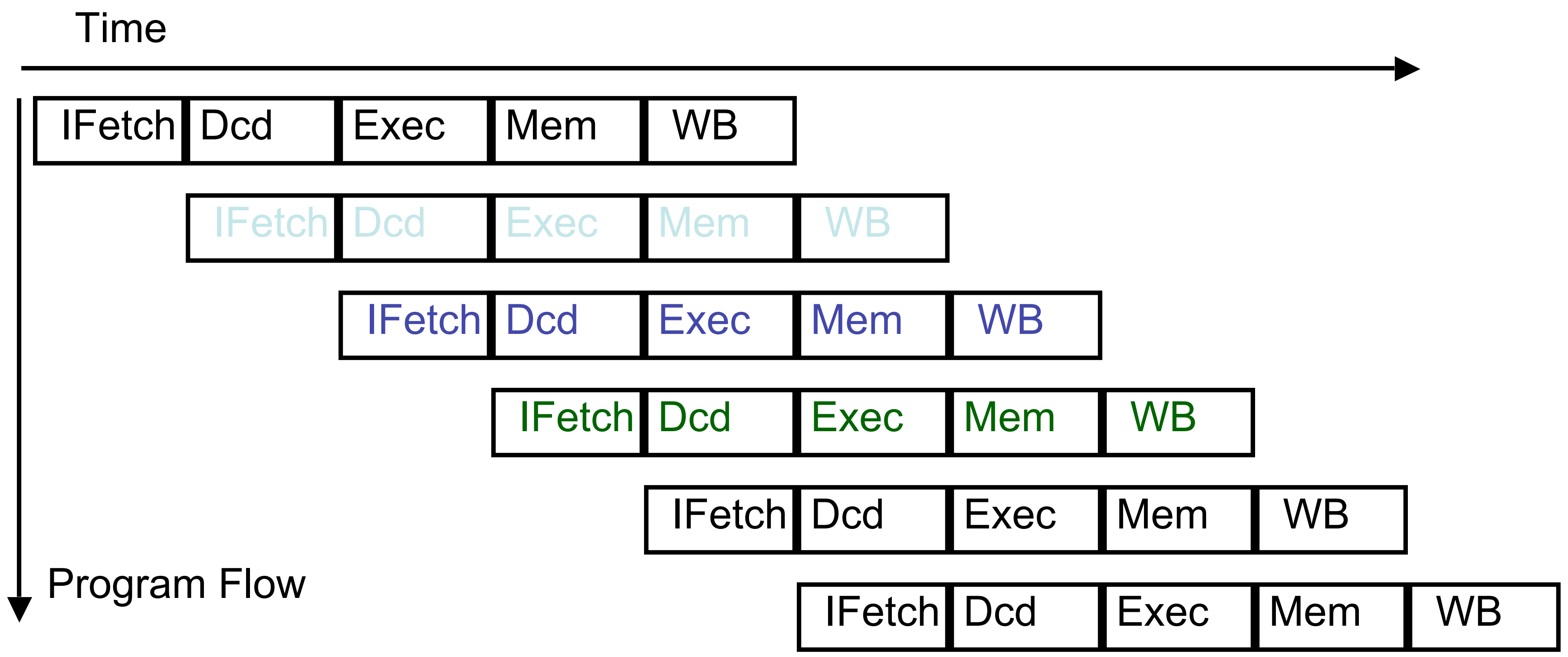


- Pipe Registers
  - Inserted between stages
  - Labeled by preceding & following stage

Pipelining: Use Multiple Resources



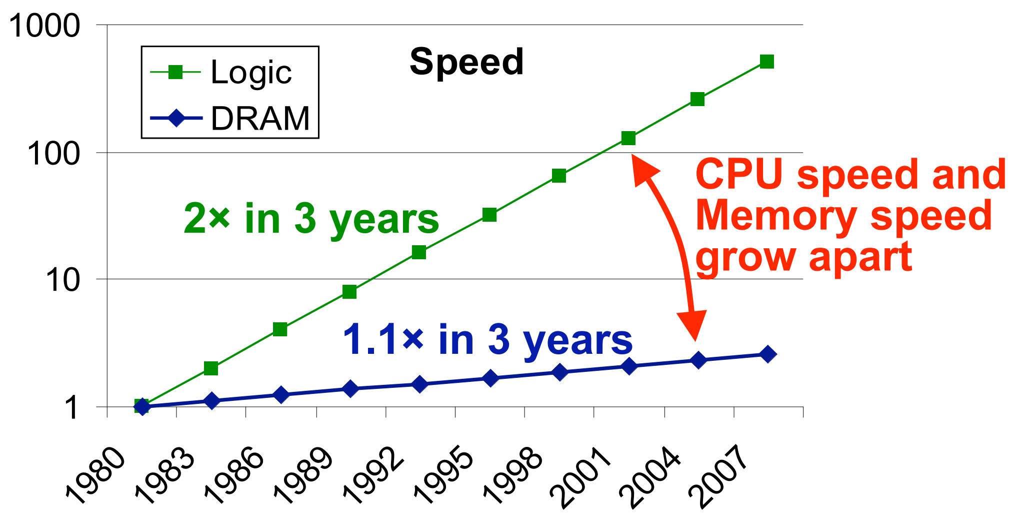
Conventional Pipelined Execution Representation

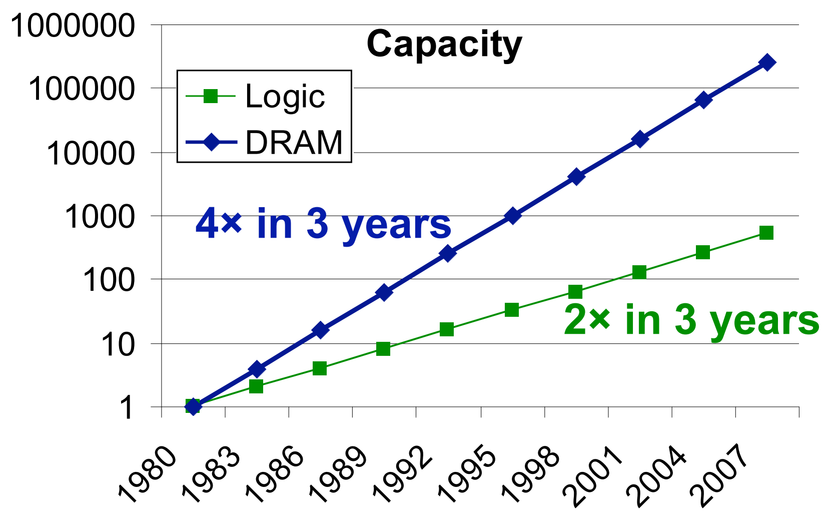


- Improve performance by increasing instruction throughput

  Ideal speedup? N Stages and I instructions

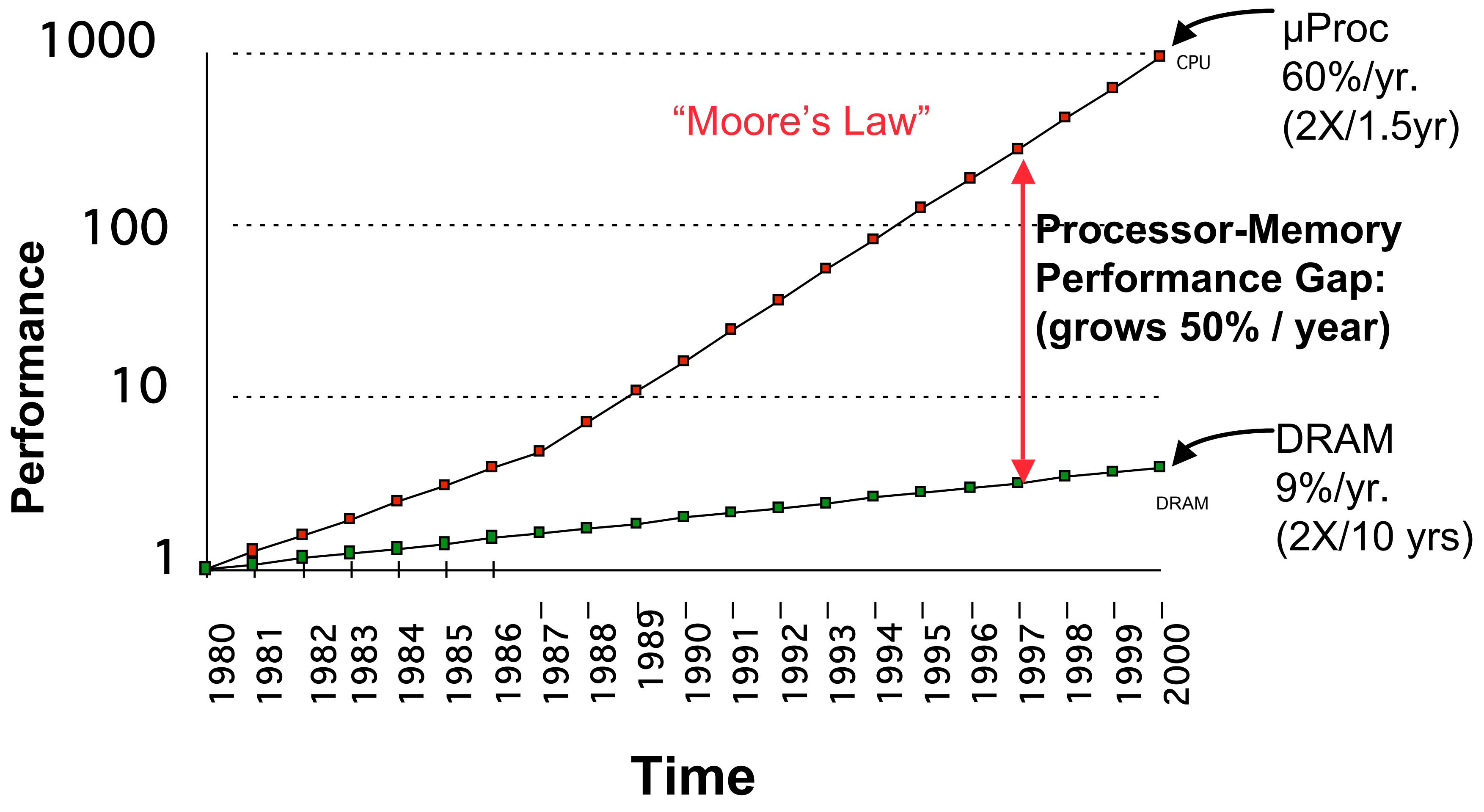
Technology Trends and Performance





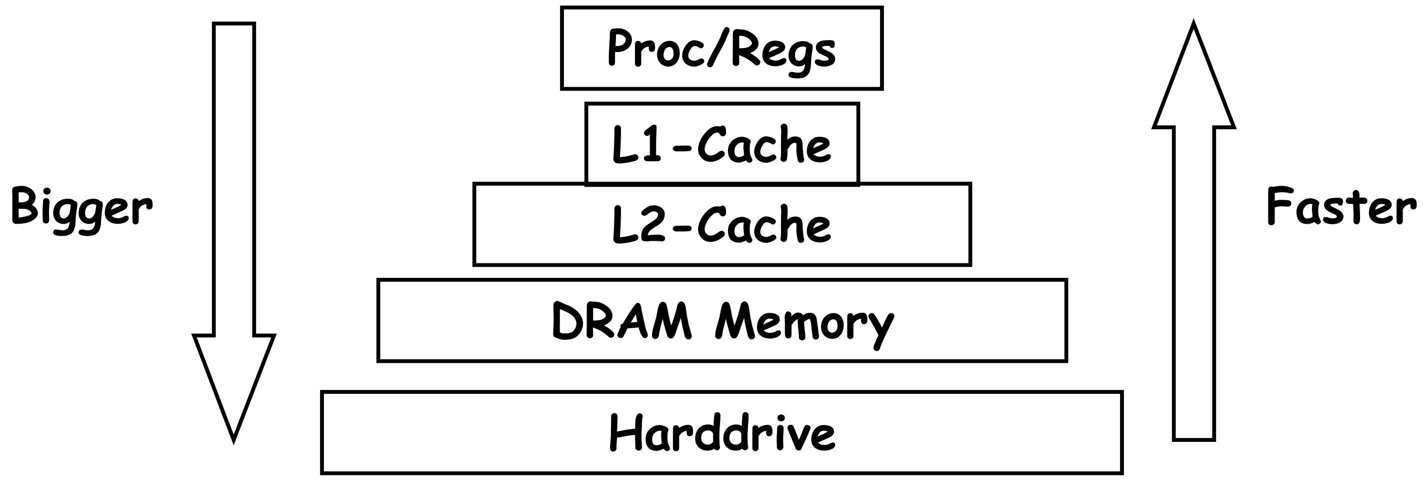
- Computing capacity: 4× per 3 years
  - If we could keep all the transistors busy all the time
  - Actual: 3.3× per 3 years
- Moore's Law: Performance is doubled every ~18 months
  - Trend is slowing: process scaling declines, power is up

Processor-DRAM Memory Gap (latency)



Cache Memory

- Small, fast storage used to improve average access time to slow memory.
- Exploits spatial and temporal locality
- In computer architecture, almost everything is a cache!
  - Registers a cache on variables
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)



Modern Processor

This is an

AMD Operton CPU

Total Area: 193 mm^2

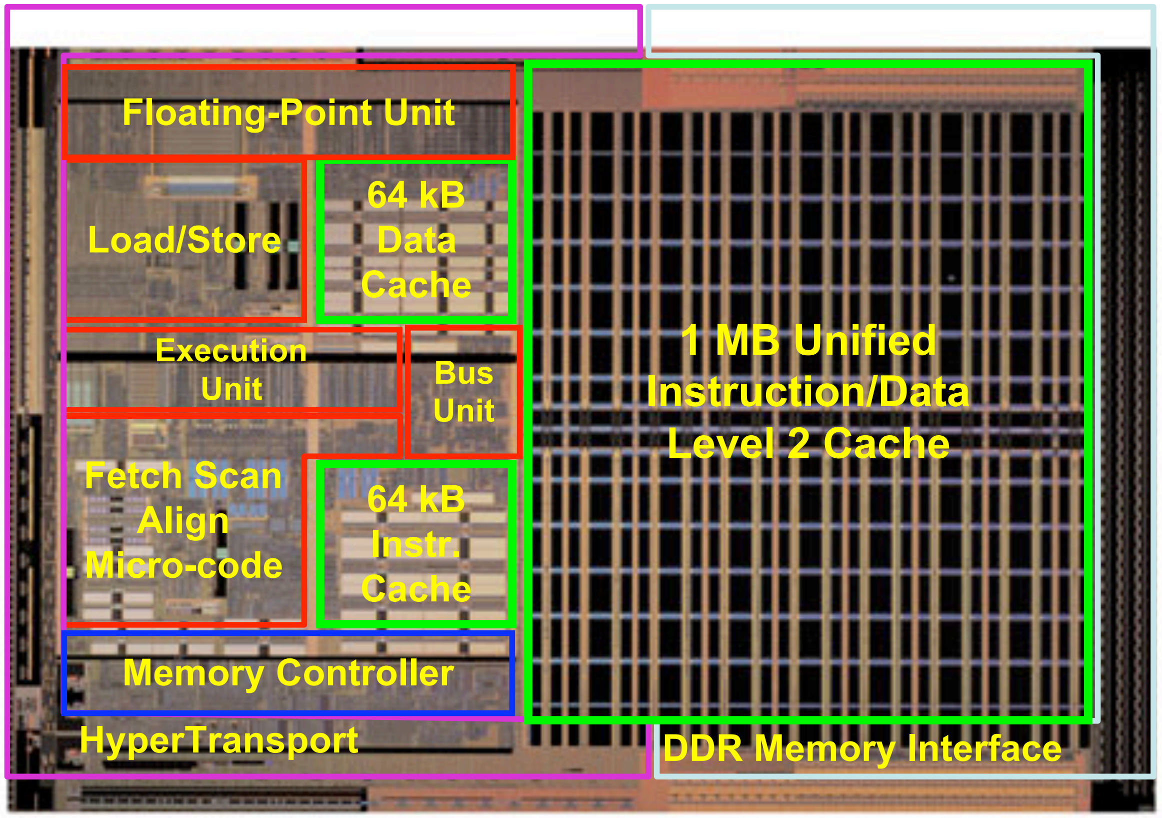
Look at the relative sizes of each block:

50% cache

23% I/O

20% CPU logic

+ extra stuff



CPUs dedicate > 50% area for cache memory.

CPI – Cycles Per Instruction

- CPUs work according to a clock signal
  - Clock cycle is measured in nsec (10^{-9} of a second)
  - Clock frequency (= 1/clock cycle) measured in GHz (10^{9}cyc/sec)
- Instruction Count (IC)
  - Total number of instructions executed in the program

$$\text{CPI} = \frac{\text{\#cycles required to execute the program}}{\text{IC}}$$

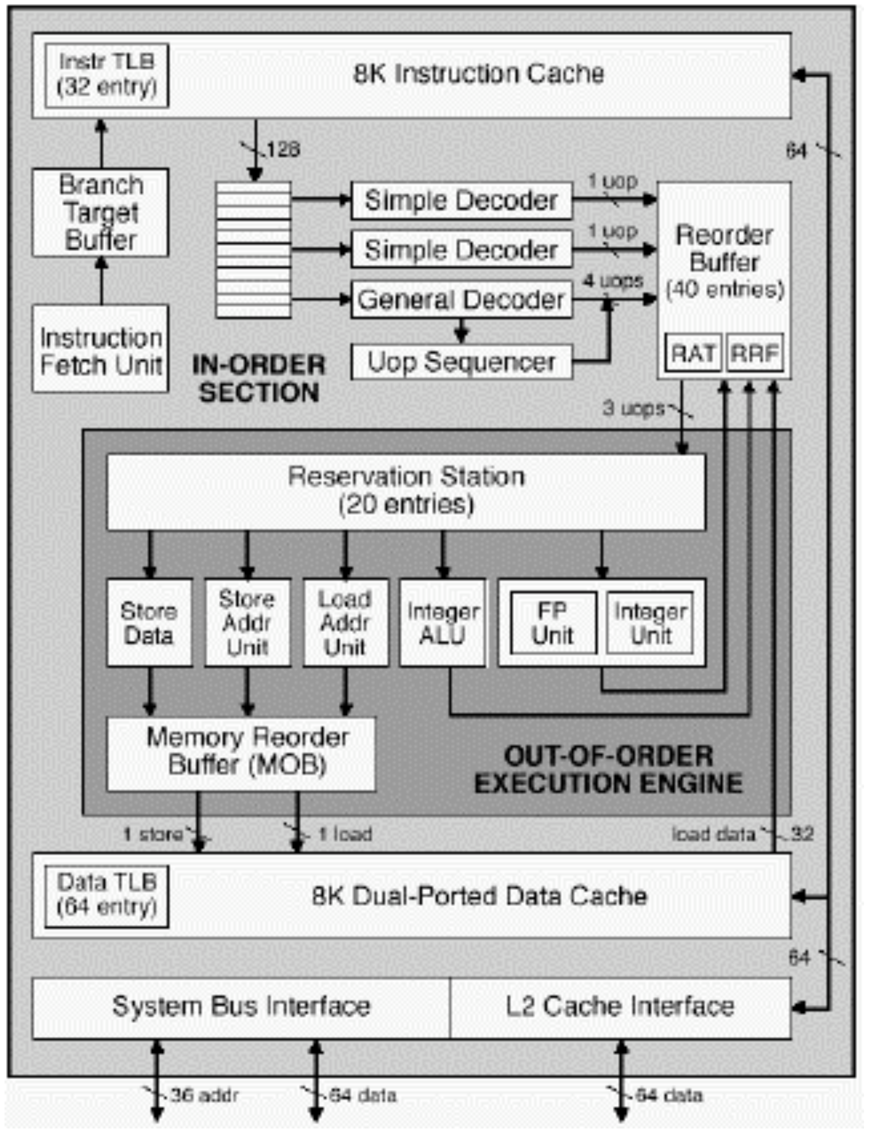
- CPI – Cycles Per Instruction
  - Average #cycles per Instruction (in a given program)
  - IPC (= 1/CPI) : Instructions per cycles

Relating time to system measures

- Iron Law: Performance = 1/execution time
- Suppose that for some program we have:
  - T seconds running time (the ultimate performance measure)
  - C clock ticks, I instructions, P seconds/tick (performance measures of interest to the system designer)
- T secs = C ticks x P secs/tick

  = (I inst/I inst) x C ticks x P secs/tick
- $$\underbrace{\text{T}}_{\text{running time}} \text{ secs} = \text{I } \underbrace{\text{inst}}_{\text{instruction count}} \text{ x } \underbrace{\text{(C ticks/I inst)}}_{\text{avg clock ticks per instruction (CPI)}} \text{ x } \underbrace{\text{P secs/tick}}_{\text{clock period}}$$

Pentium 4 Block Diagram



Microprocessor Report

Out Of Order Execution

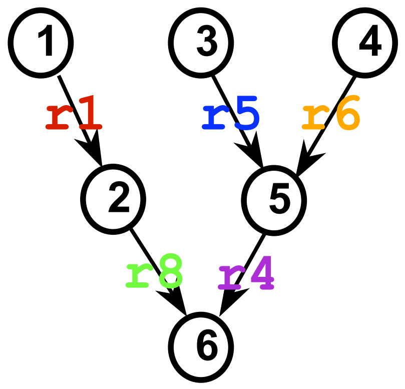
- Look ahead in a window of instructions and find instructions that are *ready to execute*
  - Don’t depend on data from previous instructions still not executed
  - Resources are available
- Out-of-order execution
  - Start instruction execution before execution of a previous instructions
- Advantages:
  - Help exploit Instruction Level Parallelism (ILP)
  - Help cover latencies (e.g., L1 data cache miss, divide)

Data Flow Analysis

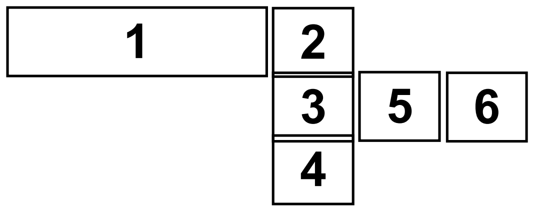
- Example:

```
(1) r1 ← r4 / r7  ; assume divide takes 20 cycles
(2) r8 ← r1 + r2
(3) r5 ← r5 + 1
(4) r6 ← r6 - r3
(5) r4 ← r5 + r6
(6) r7 ← r8 * r4
```

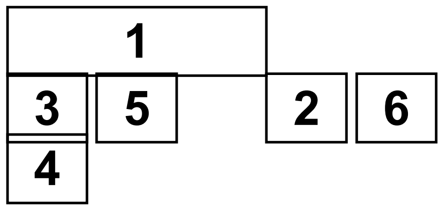
Data Flow Graph



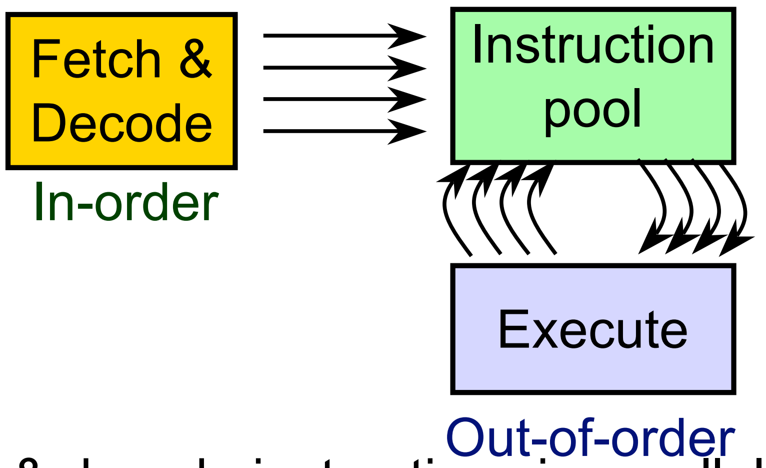
In-order execution



Out-of-order execution



OOOE – General Scheme

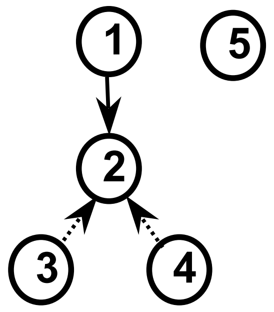


- Fetch & decode instructions in parallel but in order, to fill inst. pool
- Execute ready instructions from the instructions pool
  - All the data required for the instruction is ready
  - Execution resources are available
- Once an instruction is executed
  - signal all dependant instructions that data is ready
- Commit instructions in parallel but in-order
  - Can commit an instruction only after all preceding instructions (in program order) have committed

Out Of Order Execution – Example

- Assume that executing a divide operation takes 20 cycles

```
(1)    r1 ← r5 / r4
(2)    r3 ← r1 + r8
(3)    r8 ← r5 + 1
(4)    r3 ← r7 - 2
(5)    r6 ← r6 + r7
```



- Inst2 has a REAL (flow) dependency on r1 with Inst1
  - It cannot be executed in parallel with Inst1
- Can successive instructions pass Inst2 ?
  - Inst3 cannot since Inst2 must read r8 before Inst3 writes to it
  - Inst4 cannot since it must write to r3 after Inst2
  - Inst5 can

Overcoming False Dependencies

- OOOE creates new dependencies
  - **WAR (write after read)**: write to a register which is read by an earlier inst.

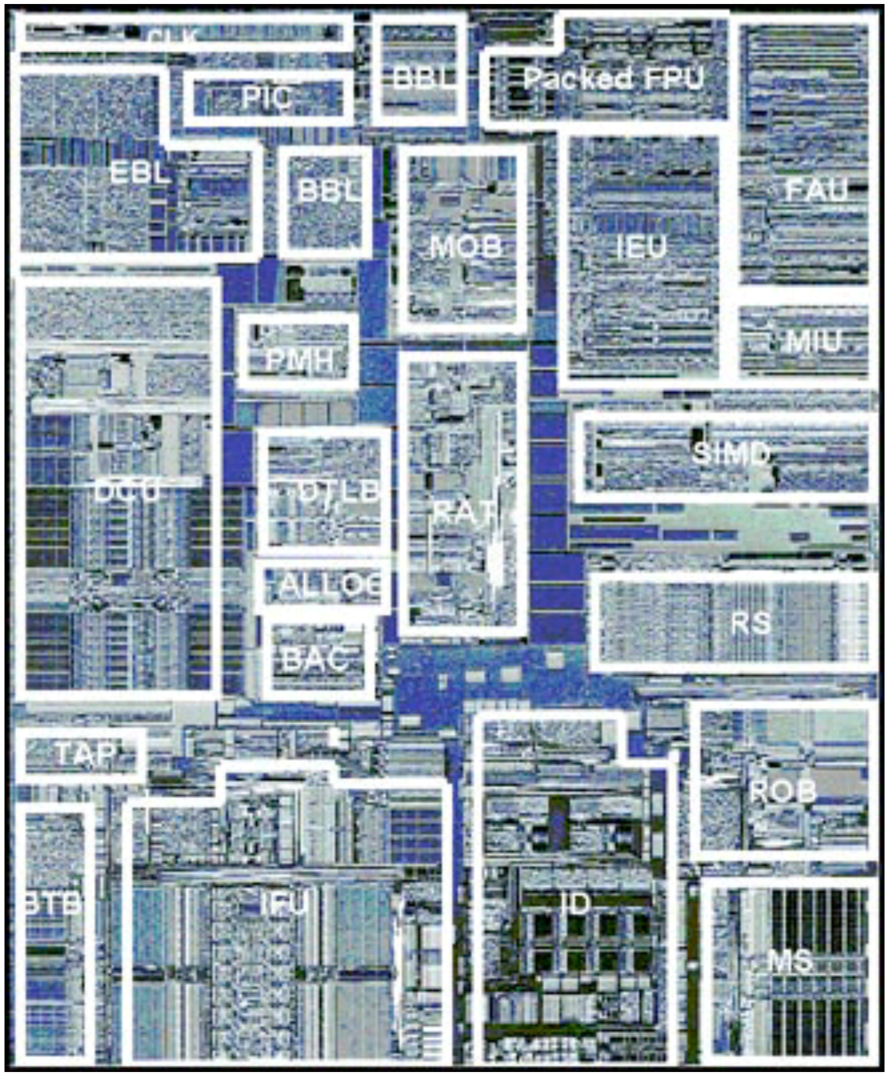
```
(1)   r3 ← r2 + r1
(2)   r2 ← r4 + 3
```

  - **WAW (write after write)**: write to a register which is written by an earlier inst.

```
(1)   r3 ← r1 + r2
(2)   r3 ← r4 + 3
```

- These are *false dependencies*
  - There is no missing data
  - Still prevent executing instructions out-of-order
- Solution: *Register Renaming*

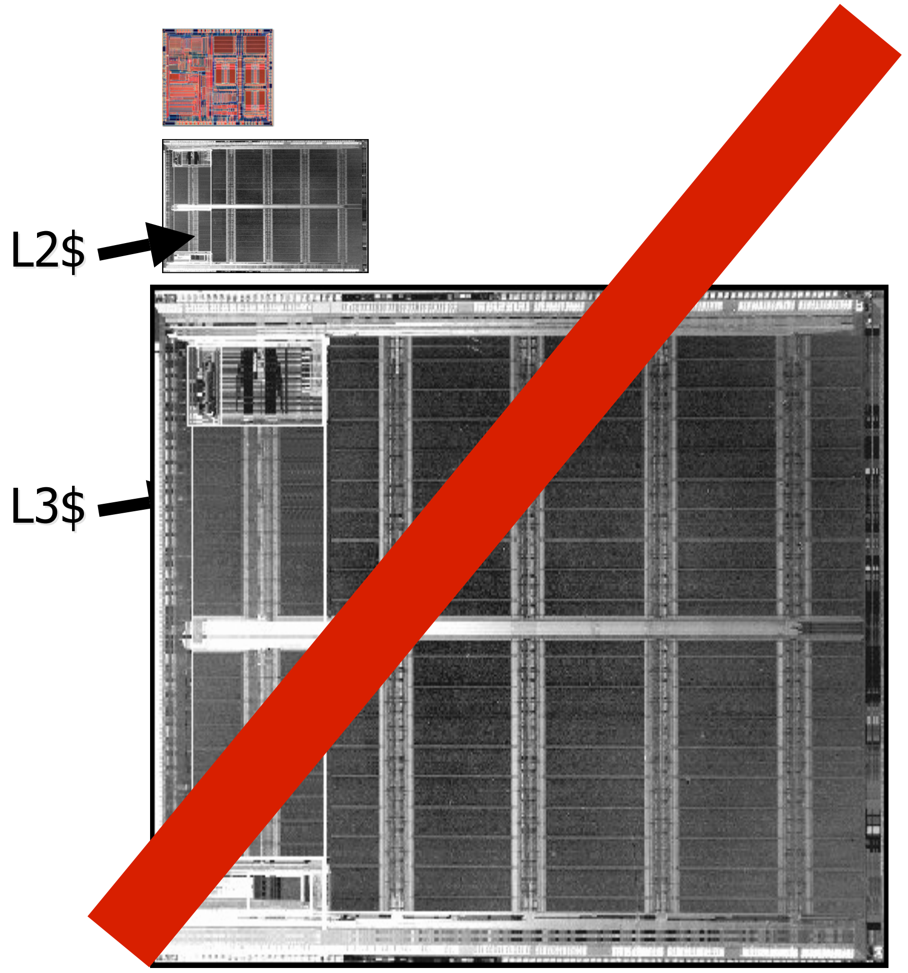
Pentium-4 Die Photo



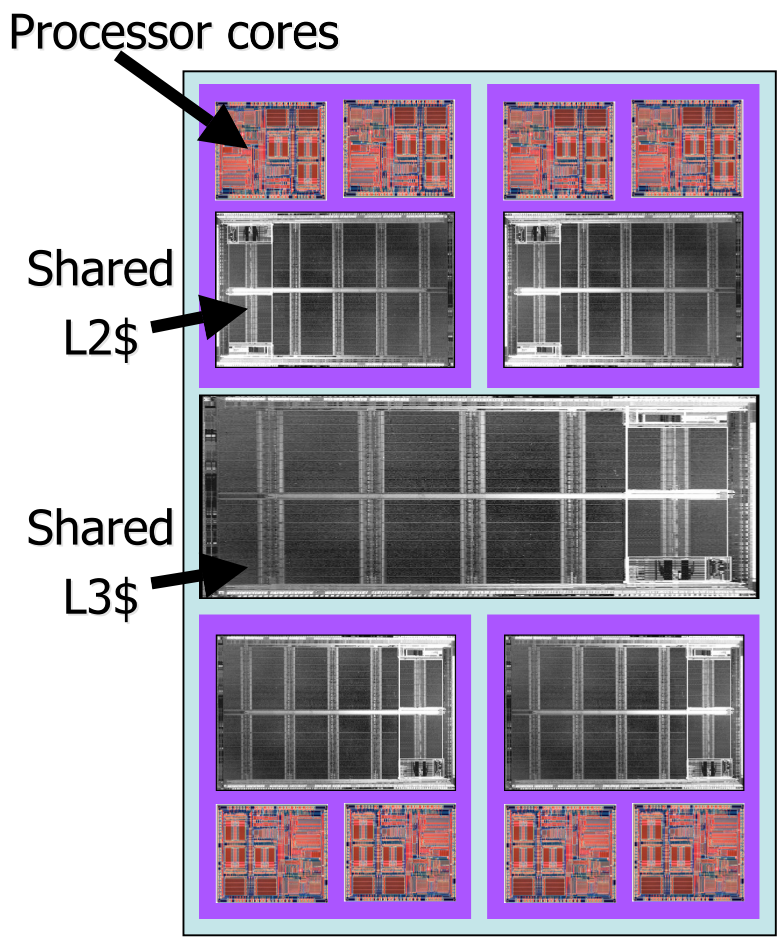
1st Pentium4: 9.5 M transistors

- EBL/BBL - Bus logic, Front, Back
- MOB - Memory Order Buffer
- Packed FPU - MMX Fl. Pt. (SSE)
- IEU - Integer Execution Unit
- FAU - Fl. Pt. Arithmetic Unit
- MIU - Memory Interface Unit
- DCU - Data Cache Unit
- PMH - Page Miss Handler
- DTLB - Data TLB
- BAC - Branch Address Calculator
- RAT - Register Alias Table
- SIMD - Packed Fl. Pt.
- RS - Reservation Station
- BTB - Branch Target Buffer
- IFU - Instruction Fetch Unit (+I$)
- ID - Instruction Decode
- ROB - Reorder Buffer
- MS - Micro-instruction Sequencer

An Exciting Time for CE - CMPs

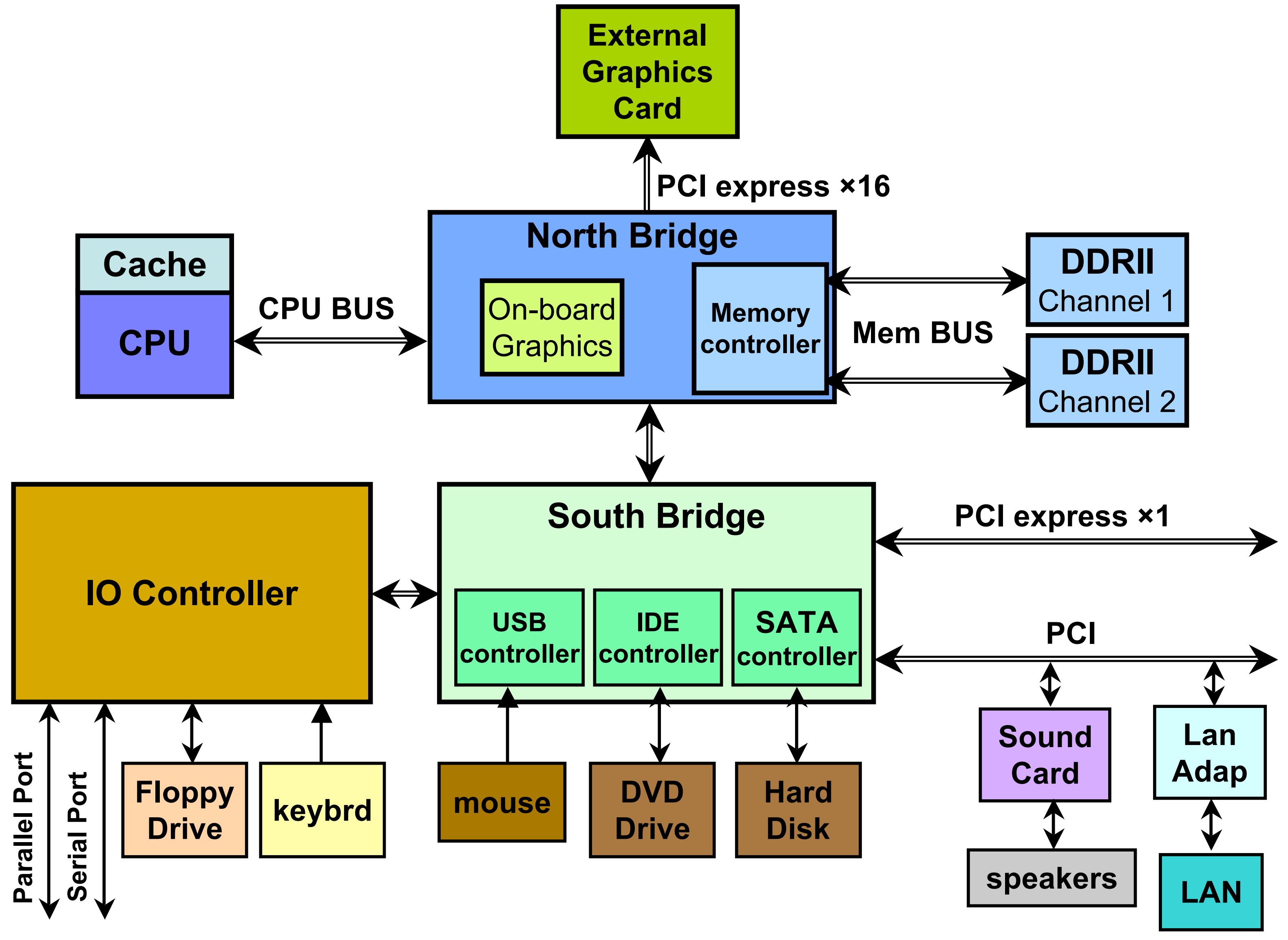


Monolithic uniprocessor



Chip multi-processor

Computer System Structure


External Graphics Card
PCI express ×16
North Bridge
On-board Graphics
Memory controller
Cache
CPU
CPU BUS
DDRII Channel 1
Mem BUS
DDRII Channel 2
South Bridge
PCI express ×1
IO Controller
USB controller
IDE controller
SATA controller
PCI
Parallel Port
Serial Port
Floppy Drive
keybrd
mouse
DVD Drive
Hard Disk
Sound Card
Lan Adap
speakers
LAN