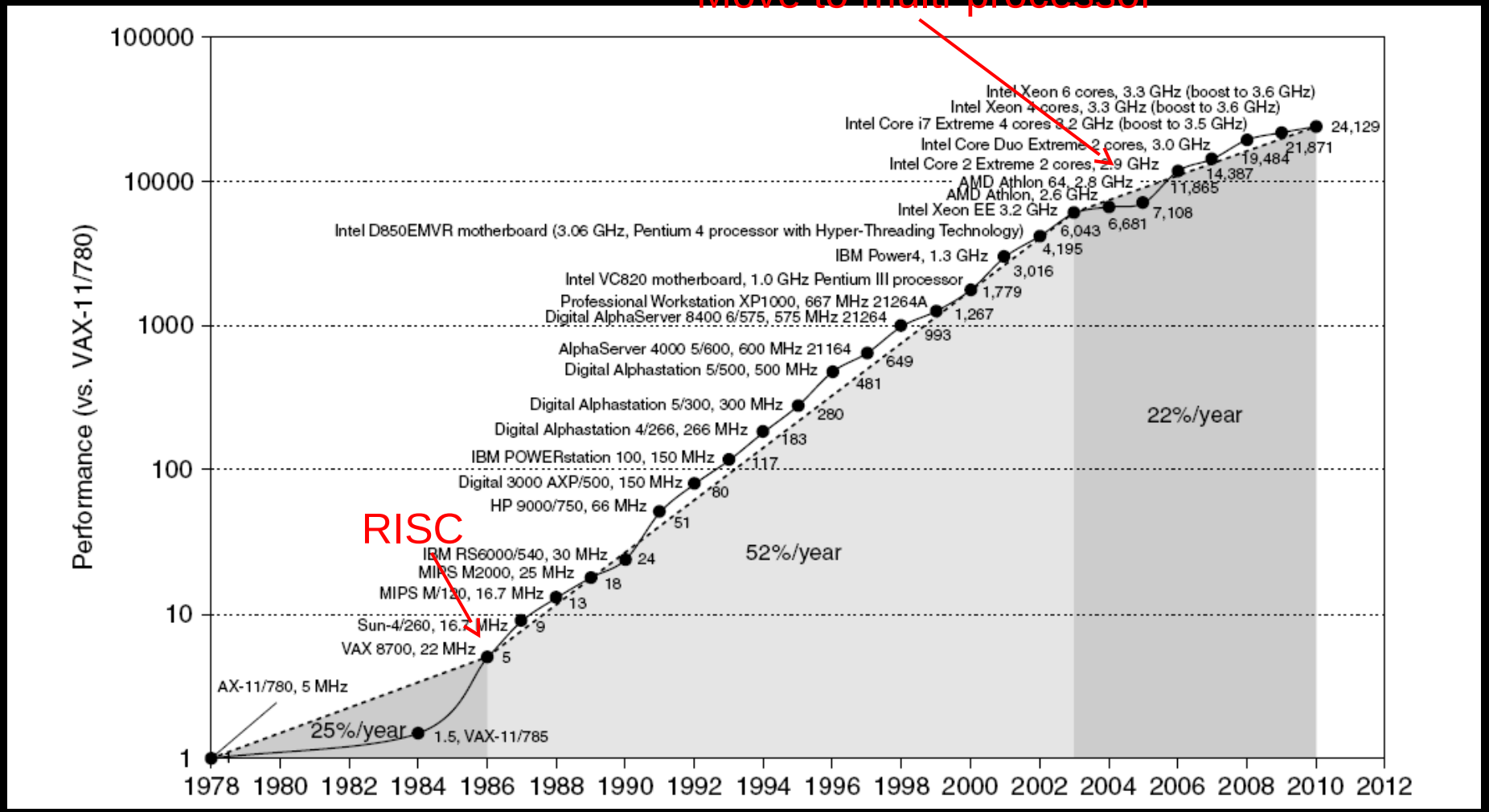
RISC, CISC, and ISA Variations

Single Processor Performance



Current Trends in Architecture

Cannot continue to leverage Instruction-Level parallelism (ILP)

- Single processor performance improvement ended in 2003

New models for performance:

- Data-level parallelism (DLP)
- Thread-level parallelism (TLP)
- Request-level parallelism (RLP)

These require explicit restructuring of the application

Parallelism

Classes of parallelism in applications:

- Data-Level Parallelism (DLP)
- Task-Level Parallelism (TLP)

Classes of architectural parallelism:

- Instruction-Level Parallelism (ILP)
- Vector architectures/Graphic Processor Units (GPUs)
- Thread-Level Parallelism
- Request-Level Parallelism

Defining Computer Architecture

"Old" view of computer architecture:

- Instruction Set Architecture (ISA) design
- i.e. decisions regarding:
  - registers, memory addressing, addressing modes, instruction operands, available operations, control flow instructions, instruction encoding

"Real" computer architecture:

- Specific requirements of the target machine
- Design to maximize performance within constraints: cost, power, and availability
- Includes ISA, microarchitecture, hardware

Trends in Technology

Integrated circuit technology

- Transistor density: 35%/year
- Die size: 10-20%/year
- Integration overall: 40-55%/year

DRAM capacity: 25-40%/year (slowing)

Flash capacity: 50-60%/year

- 15-20X cheaper/bit than DRAM

Magnetic disk technology: 40%/year

- 15-25X cheaper/bit then Flash
- 300-500X cheaper/bit than DRAM

Bandwidth and Latency

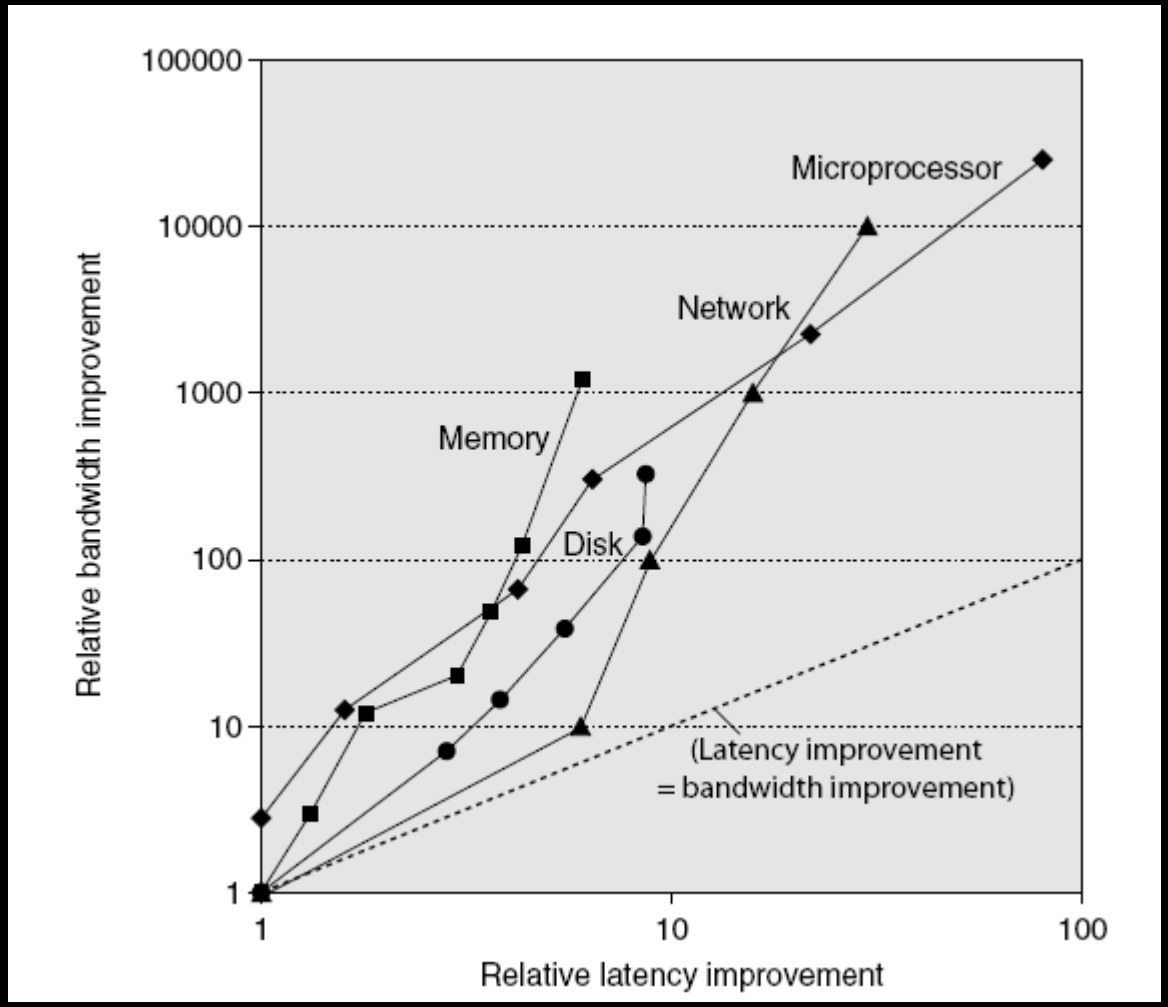
Bandwidth or throughput

- Total work done in a given time
- 10,000-25,000X improvement for processors
- 300-1200X improvement for memory and disks

Latency or response time

- Time between start and completion of an event
- 30-80X improvement for processors
- 6-8X improvement for memory and disks

Bandwidth and Latency



Log-log plot of bandwidth and latency milestones

Transistors and Wires

Feature size

- Minimum size of transistor or wire in x or y dimension
- 10 microns in 1971 to .032 microns in 2011
- Transistor performance scales linearly

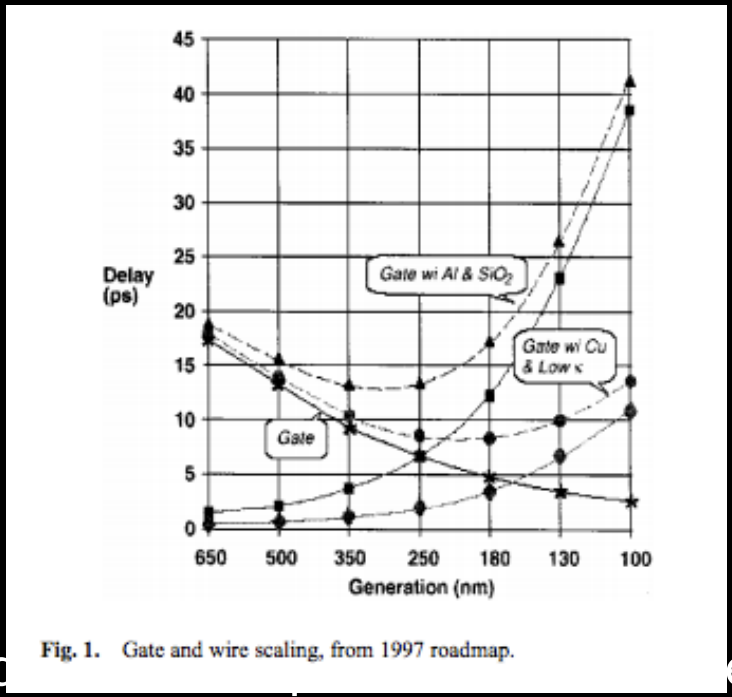


Fig. 1. Gate and wire scaling, from 1997 roadmap.

  - Wire delay d e size!
- Integration density scales quadratically

Power and Energy

Problem: Get power in, get power out

Thermal Design Power (TDP)

- Characterizes sustained power consumption
- Used as target for power supply and cooling system
- Lower than peak power, higher than average power consumption

Clock rate can be reduced dynamically to limit power consumption

Energy per task is often a better measurement

Dynamic Energy and Power

Dynamic energy

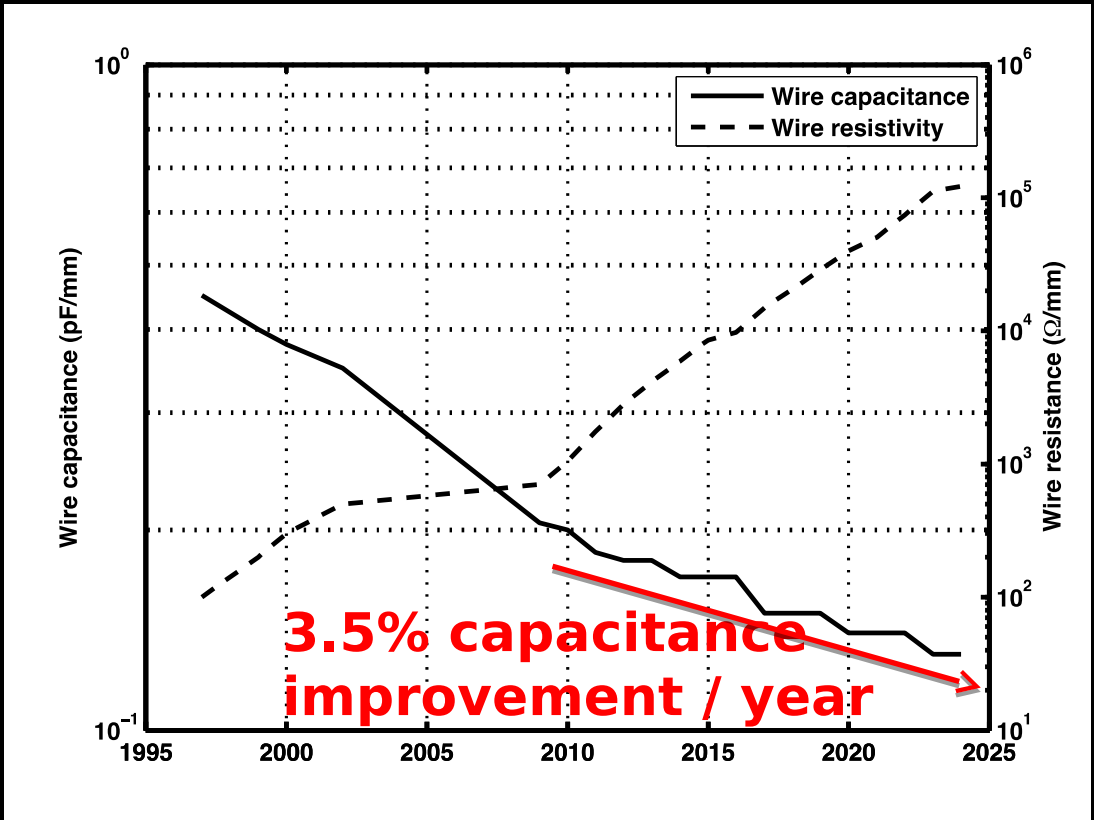
- Transistor switch from 0 -> 1 or 1 -> 0
- ½ x Capacitive load x Voltage^2

Dynamic power

- ½ x Capacitive load x Voltage^2 x Frequency switched

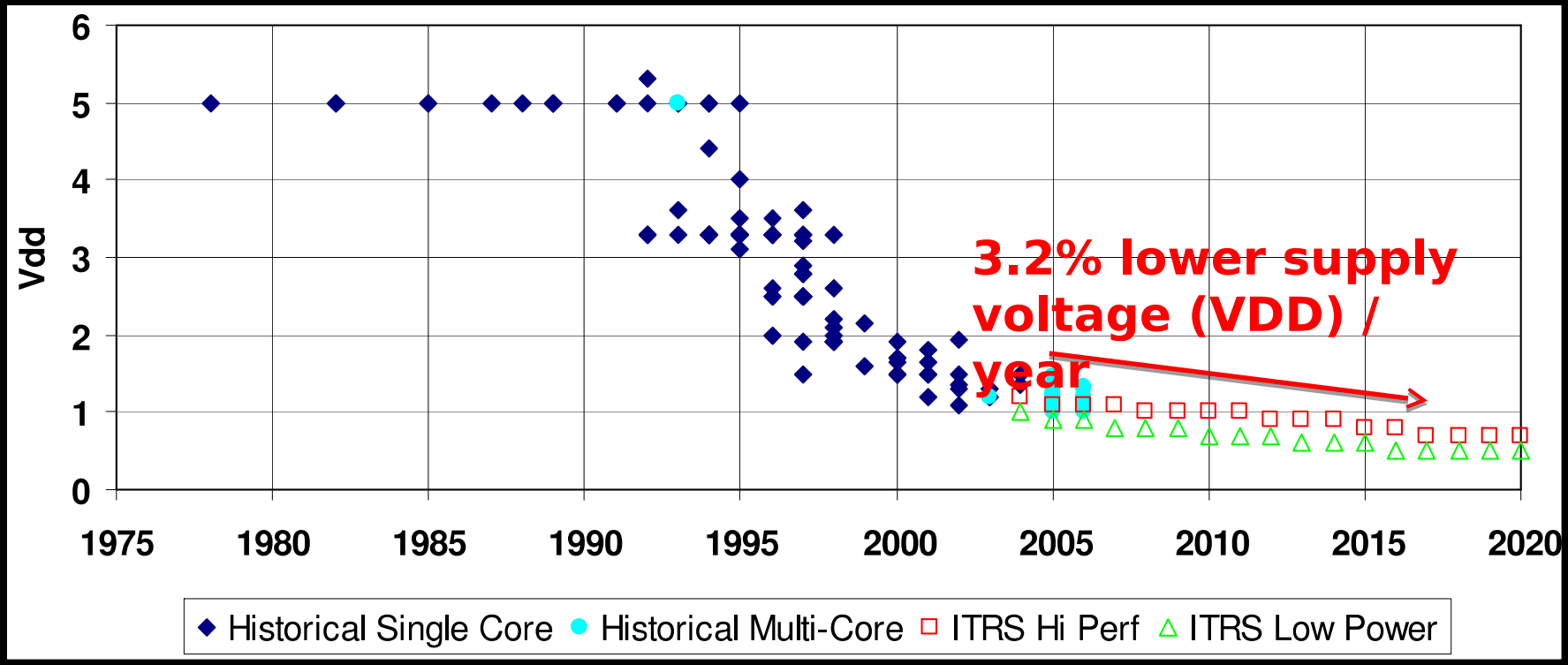
Reducing clock rate reduces power, not energy

(CV^2f) Power not scaling!



WHY DOESN'T CAPACITANCE SCALE?

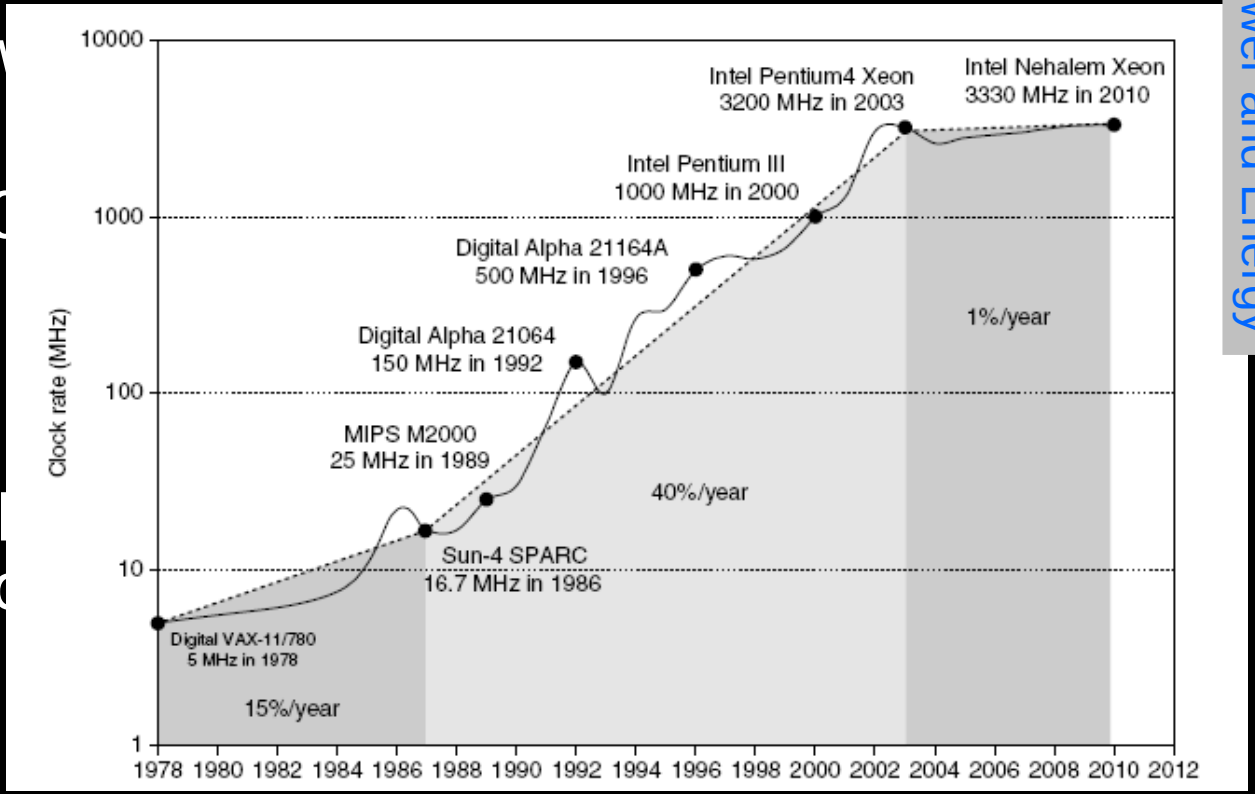
(CV^2f) Power not scaling!



Why Not?

Power

- Intel 80386 consumed ~ 2 W
- 3.3 GHz Intel Co i7 consumes 130 W
- Heat must be dissipated from 1.5 x 1.5 cm chi
- This is the limit o what can be cooled by air

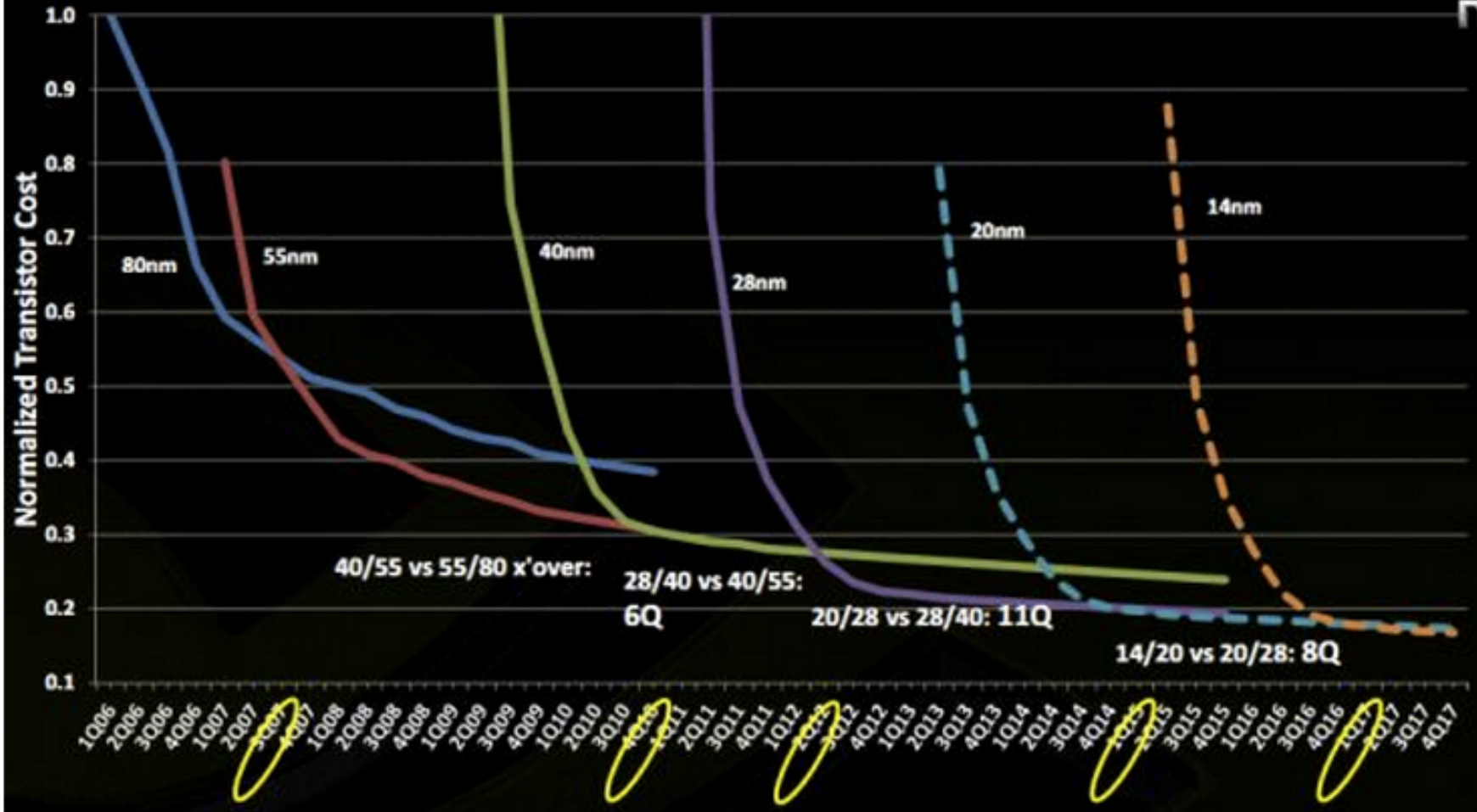


Reducing Power

Techniques for reducing power:

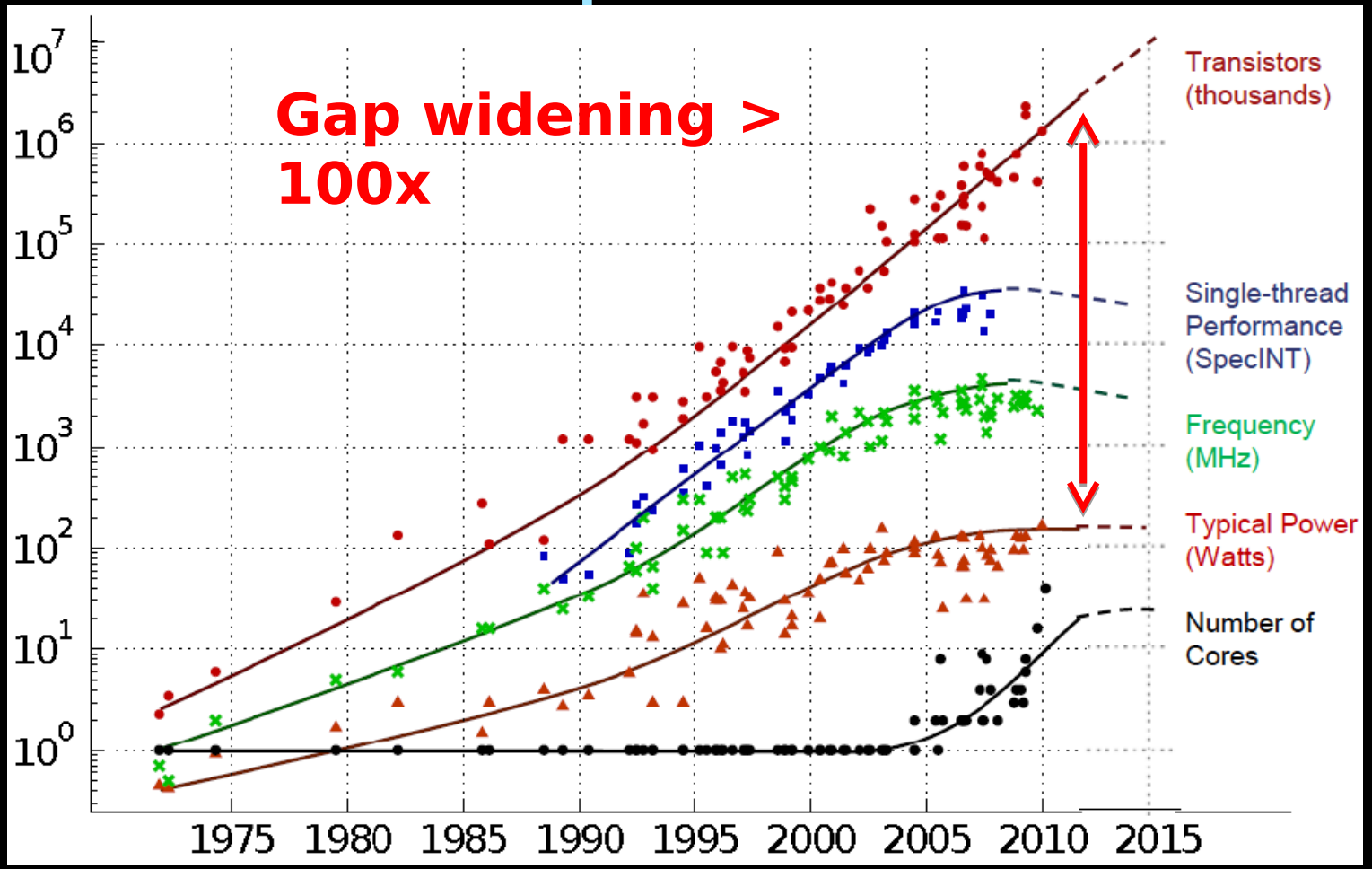
- Do nothing well
- Dynamic Voltage-Frequency Scaling
- Low power state for DRAM, disks
- Overclocking, turning off cores

Pricing: X'over on Transistor Cost



- *X'tor cost = F (yield(t), scaling factor, wafer cost)*
- *X'over not quite on the 2yr (8Q) cadence.*
- *20 or 14nm cost barely goes below the previous one, no saving!*

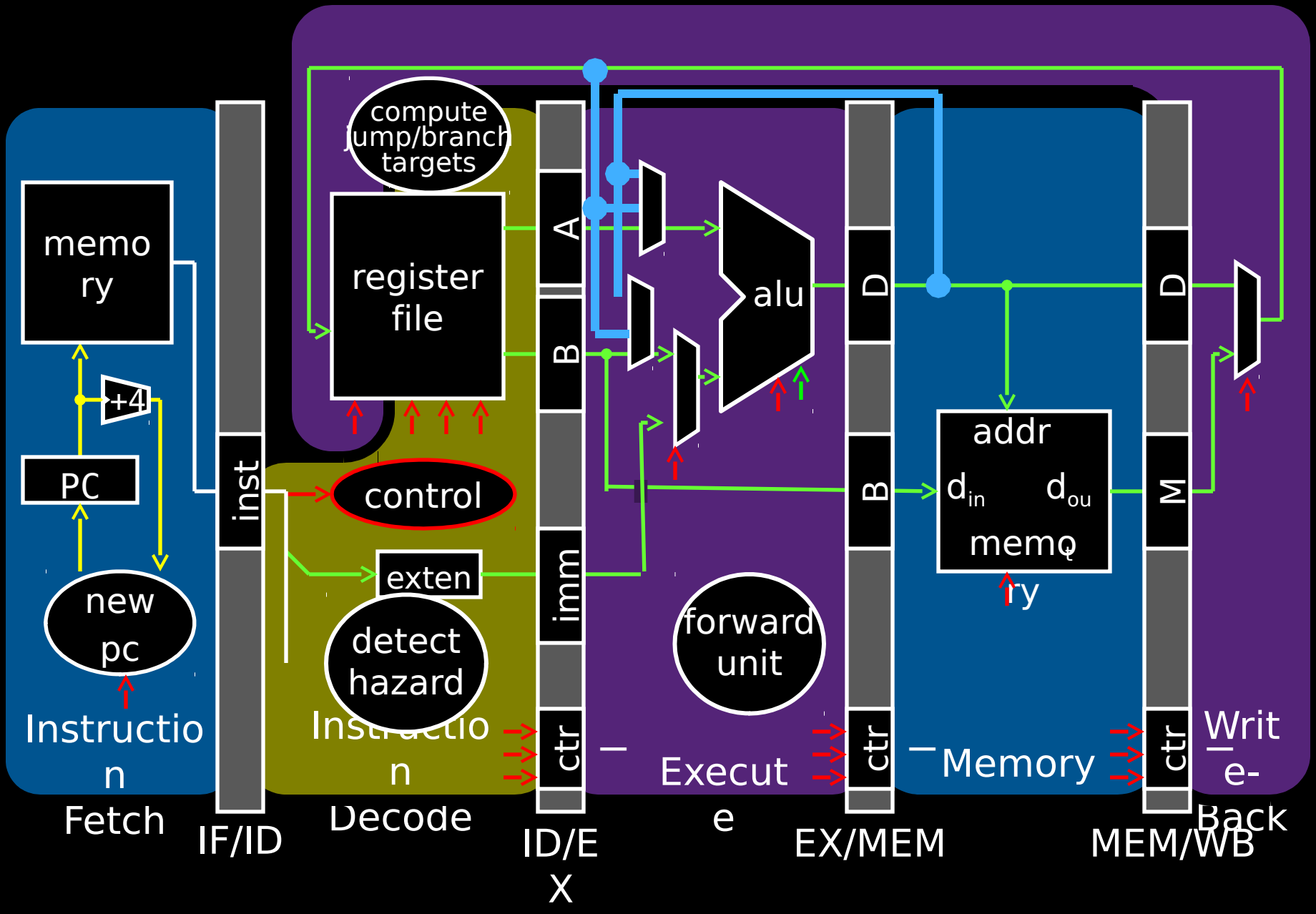
More transistors than you have power



‘DARK SILICON’

[2]: Chuck Moore, AMD

Big Picture: Where are we



Big Picture: Where are we going?

C

compiler

MIPS assembly

assembler

machine code

CPU

Circuits

Gates

Transistors

Silicon

```
int x = 10;
x = 2 * x + 15;
```

```
addi r5, r0, 10
muli r5, r5, 2
addi r5, r5, 15
```

r0 = 0

r5 = r0 + 10

r5 = r5<<1 #r5 = r5 *

r5 = r15 + 15

op = addi r0 r5 10

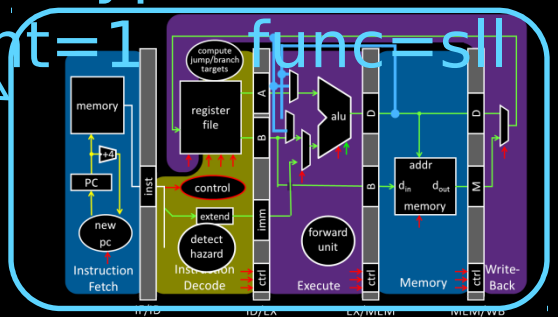
```
00100000000001010000000000001010
```

```
00000000000001010010100001000000
```

op = addi r5 r5 15

```
00100000101001010000000000001111
```

op = r-type r5 r5 shamt=1 func=sll



Big Picture: Where are we going?

C

compiler

MIPS assembly

assembler

machine code

CPU

Circuits

Gates

Transistors

Silicon

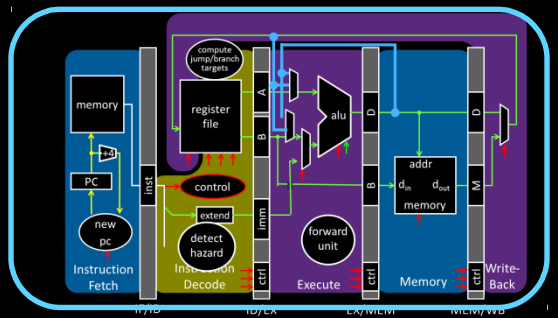
```
int x = 10;
x = 2 * x + 15;
```

```
addi r5, r0, 10
muli r5, r5, 2
addi r5, r5, 15
```

```
00100000000001010000000000001010
00000000000010100101010000010000
00100000101001010000000000000111
```

High Level Languages

Instruction Set Architecture (ISA)



Instruction Set Architecture Variations

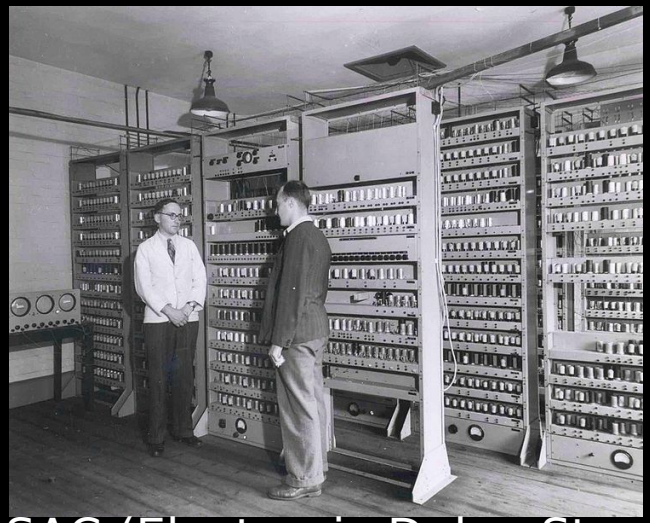
ISA defines the permissible instructions

- MIPS: load/store, arithmetic, control flow, ...
- ARMv7: similar to MIPS, but more shift, memory, & conditional ops
- ARMv8 (64-bit): even closer to MIPS, no conditional ops
- VAX: arithmetic on memory or registers, strings, polynomial evaluation, stacks/queues, ...
- Cray: vector operations, ...
- x86: a little of everything

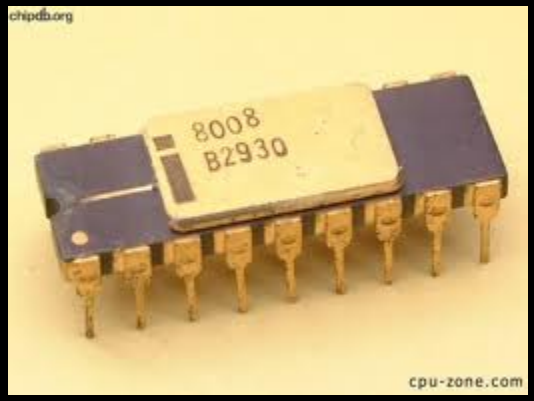
Brief Historical Perspective on ISAs

Accumulators

- Early stored-program computers had ***one*** register!



EDSAC (Electronic Delay Storage Automatic Calculator) in 1949



Intel 8008 in 1972 was an accumulator

- One register is two registers short of a MIPS instruction!
- Requires a memory-based operand-addressing mode
  - Example Instructions: add 200
    - Add the accumulator to the word in memory at address 200
    - Place the sum back in the accumulator

Brief Historical Perspective on ISAs

Next step, more registers…

- Dedicated registers
  - E.g. indices for array references in data transfer instructions, separate accumulators for multiply or divide instructions, top-of-stack pointer.

Intel 8086
“extended accumulator”
Processor for IBM PCs

- Extended Accumulator
  - One operand may be in memory (like previous accumulators).
  - Or, all the operands may be registers (like MIPS).

Brief Historical Perspective on ISAs

Next step, more registers…

- General-purpose registers
  - Registers can be used for any purpose
  - E.g. MIPS, ARM, x86

- *Register-memory* architectures
  - One operand may be in memory (e.g. accumulators)
  - E.g. x86 (i.e. 80386 processors
- *Register-register* architectures (aka load-store)
  - All operands ***must*** be in registers
  - E.g. MIPS, ARM

Takeaway

The number of available registers greatly influenced the instruction set architecture (ISA)

Machine	Num General Purpose Registers	Architectural Style	Year
EDSAC	1	Accumulator	1949
IBM 701	1	Accumulator	1953
CDC 6600	8	Load-Store	1963
IBM 360	18	Register-Memory	1964
DEC PDP-8	1	Accumulator	1965
DEC PDP-11	8	Register-Memory	1970
Intel 8008	1	Accumulator	1972
Motorola 6800	2	Accumulator	1974
DEC VAX	16	Register-Memory, Memory-Memory	1977
Intel 8086	1	Extended Accumulator	1978
Motorola 6800	16	Register-Memory	1980
Intel 80386	8	Register-Memory	1985
ARM	16	Load-Store	1985
MIPS	32	Load-Store	1985
HP PA-RISC	32	Load-Store	1986
SPARC	32	Load-Store	1987
PowerPC	32	Load-Store	1992
DEC Alpha	32	Load-Store	1992

Takeaway

The number of available registers greatly influenced the instruction set architecture (ISA)

Machine	Number of general-purpose registers	Architectural style	Year
EDSAC	1	Accumulator	1949
IBM 701	1	Accumulator	1953
CDC 6600	8	Load-store	1963
IBM 360	16	Register-memory	1964
DEC PDP-8	1	Accumulator	1965
DEC PDP-11	8	Register-memory	1970
Intel 8008	1	Accumulator	1972
Motorola 6800	2	Accumulator	1974
DEC VAX	16	Register-memory, memory-memory	1977
Intel 8086	1	Extended accumulator	1978
Motorola 68000	16	Register-memory	1980
Intel 80386	8	Register-memory	1985
ARM	16	Load-store	1985
MIPS	32	Load-store	1985
HP PA-RISC	32	Load-store	1986
SPARC	32	Load-store	1987
PowerPC	32	Load-store	1992
DEC Alpha	32	Load-store	1992
HP/Intel IA-64	128	Load-store	2001
AMD64 (EMT64)	16	Register-memory	2003

People programmed in assembly and machine code!

- Needed as many addressing modes as possible
- Memory was (and still is) slow

CPUs had relatively few registers

- Register’s were more “expensive” than external mem
- Large number of registers requires many bits to index

Memories were small

- Encouraged highly encoded microcodes as instructions
- Variable length instructions, load/store, conditions, etc

People programmed in assembly and machine code!

E.g. x86

- > 1000 instructions!
  - 1 to 15 bytes each
  - E.g. dozens of add instructions
- operands in dedicated registers, general purpose registers, memory, on stack, …
  - can be 1, 2, 4, 8 bytes, signed or unsigned
- 10s of addressing modes
  - e.g. Mem[segment + reg + reg*scale + offset]

E.g. VAX

- Like x86, arithmetic on memory or registers, but also on strings, polynomial evaluation, stacks/queues

Complex Instruction Set Computers (CISC)

Takeaway

The number of available registers greatly influenced the instruction set architecture (ISA)

Complex Instruction Set Computers were very complex

- Necessary to reduce the number of instructions required to fit a program into memory.
- However, also greatly increased the complexity of the ISA as well.

Reduced Instruction Set Computer (RISC)

John Cock

- IBM 801, 1980 (started in 1975)
- Name 801 came from the bldg that housed the project
- Idea: Possible to make a very small and very fast core
- Influences: Known as “the father of RISC

Award Recipient and ence.

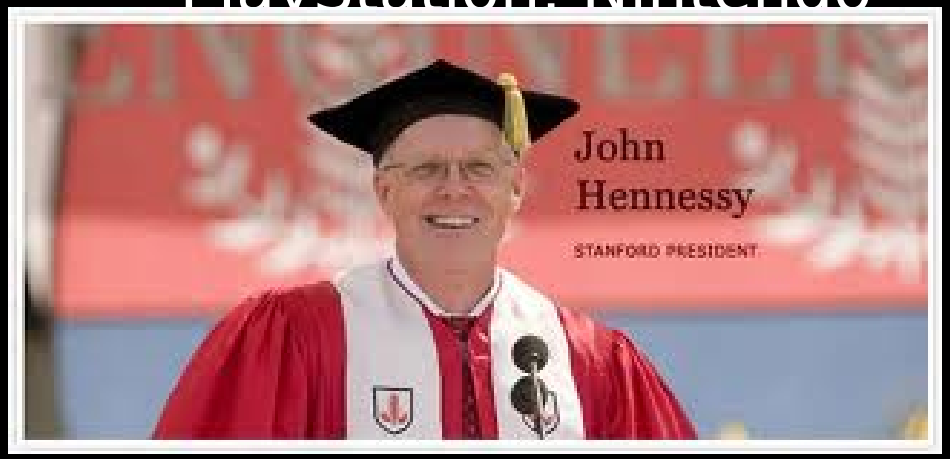
Reduced Instruction Set Computer (RISC)

Dave Patterson

- RISC Project, 1982
- UC Berkeley
- RISC-I: ½ transistors & 3x faster
- Influences: Sun SPARC, namesake of industry

John L. Hennessy

- MIPS, 1981
- Stanford
- Simple pipelining, keep full
- Influences: MIPS computer system, PlayStation, Nintendo



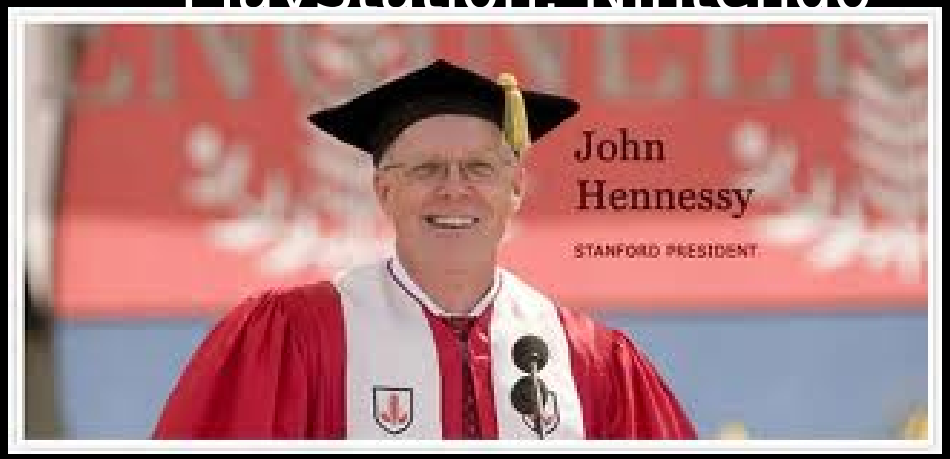
Reduced Instruction Set Computer (RISC)

Dave Patterson

- RISC Project, 1982
- UC Berkeley
- RISC-I: ½ transistors & 3x faster
- Influences: Sun SPARC, namesake of industry

John L. Hennessy

- MIPS, 1981
- Stanford
- Simple pipelining, keep full
- Influences: MIPS computer system, PlayStation, Nintendo



Reduced Instruction Set Computer (RISC)

MIPS Design Principles

Simplicity favors regularity

- 32 bit instructions

Smaller is faster

- Small register file

Make the common case fast

- Include support for constants

Good design demands good compromises

- Support for different type of interpretations/classes

Reduced Instruction Set Computer

MIPS = Reduced Instruction Set Computer (RISC)

- ≈ 200 instructions, 32 bits each, 3 formats
- all operands in registers
  - almost all are 32 bits each
- ≈ 1 addressing mode: Mem[reg + imm]

x86 = Complex Instruction Set Computer (CISC)

- > 1000 instructions, 1 to 15 bytes each
- operands in dedicated registers, general purpose registers, memory, on stack, …
  - can be 1, 2, 4, 8 bytes, signed or unsigned
- 10s of addressing modes
  - e.g. Mem[segment + reg + reg*scale + offset]

RISC vs CISC

RISC Philosophy

- Regularity & simplicity
- Leaner means faster
- Optimize the common case

CISC Rebuttal

- Compilers can be smart
- Transistors are plentiful
- Legacy is important
- Code size counts
- Micro-code!

Energy efficiency

Embedded Systems

Desktops/Servers

ARMDroid vs WinTel

- Android OS on ARM processor

- Windows OS on Intel (x86) processor

Takeaway

The number of available registers greatly influenced the instruction set architecture (ISA)

Complex Instruction Set Computers were very complex

- Necessary to reduce the number of instructions required to fit a program into memory.
- However, also greatly increased the complexity of the ISA as well.

Back in the day… CISC was necessary because everybody programmed in assembly and machine code! Today, CISC ISA's are still dominant due to the prevalence of x86 ISA processors. However, RISC ISA's today such as ARM have an ever increasing market share (of our everyday life!).

ARM borrows a bit from both RISC and CISC.

ARMv7 Instruction Set Architecture

All ARMv7 instructions are 32 bits long, has 3 formats

Reduced Instruction Set Computer (RISC) properties

- Only Load/Store instructions access memory
- Instructions operate on operands in processor registers
- 16 registers

Complex Instruction Set Computer (CISC) properties

- Autoincrement, autodecrement, PC-relative addressing
- Conditional execution
- Multiple words can be accessed from memory with a single instruction (SIMD: single instr multiple data)

ARMv8 (64-bit) Instruction Set Architecture

All ARMv8 instructions are 64 bits long, has 3 formats

Reduced Instruction Set Computer (RISC) properties

- Only Load/Store instructions access memory
- Instructions operate on operands in processor registers
- 16 registers

Complex Instruction Set Computer (CISC) properties

- Autoincrement, autodecrement, PC-relative addressing
- Conditional execution
- Multiple words can be accessed from memory with a single instruction (SIMD: single instr multiple data)

Instruction Set Architecture Variations

ISA defines the permissible instructions

- MIPS: load/store, arithmetic, control flow, ...
- ARMv7: similar to MIPS, but more shift, memory, & conditional ops
- ARMv8 (64-bit): even closer to MIPS, no conditional ops
- VAX: arithmetic on memory or registers, strings, polynomial evaluation, stacks/queues, ...
- Cray: vector operations, ...
- x86: a little of everything

Conclusion – RISC vs. CISC?

CISC

- *Effectively realizes one* particular High Level Language Computer System *in HW* - recurring **HW development costs** when change needed

RISC

- *Allows effective realization of any* High Level Language Computer System *in SW* - recurring **SW development costs** when change needed

Conclusion – Optimum?

Hybrid solutions

- RISC core & CISC interface
- Still has *specific performance tuning*

Optimal ISA

- Between RISC & CISC
- Few, carefully chosen, useful complex instructions
- Still has *complexity handling problems*