CS 152
Computer Architecture and Engineering
Lecture 5 – ISA Design + Microcode + Cost

BORN ON THIS DAY IN 2004.

FACEBOOK

2014-2-4

John Lazzaro

(not a prof - "John" is always OK)

TA: Eric Love

www-inst.eecs.berkeley.edu/~cs152/

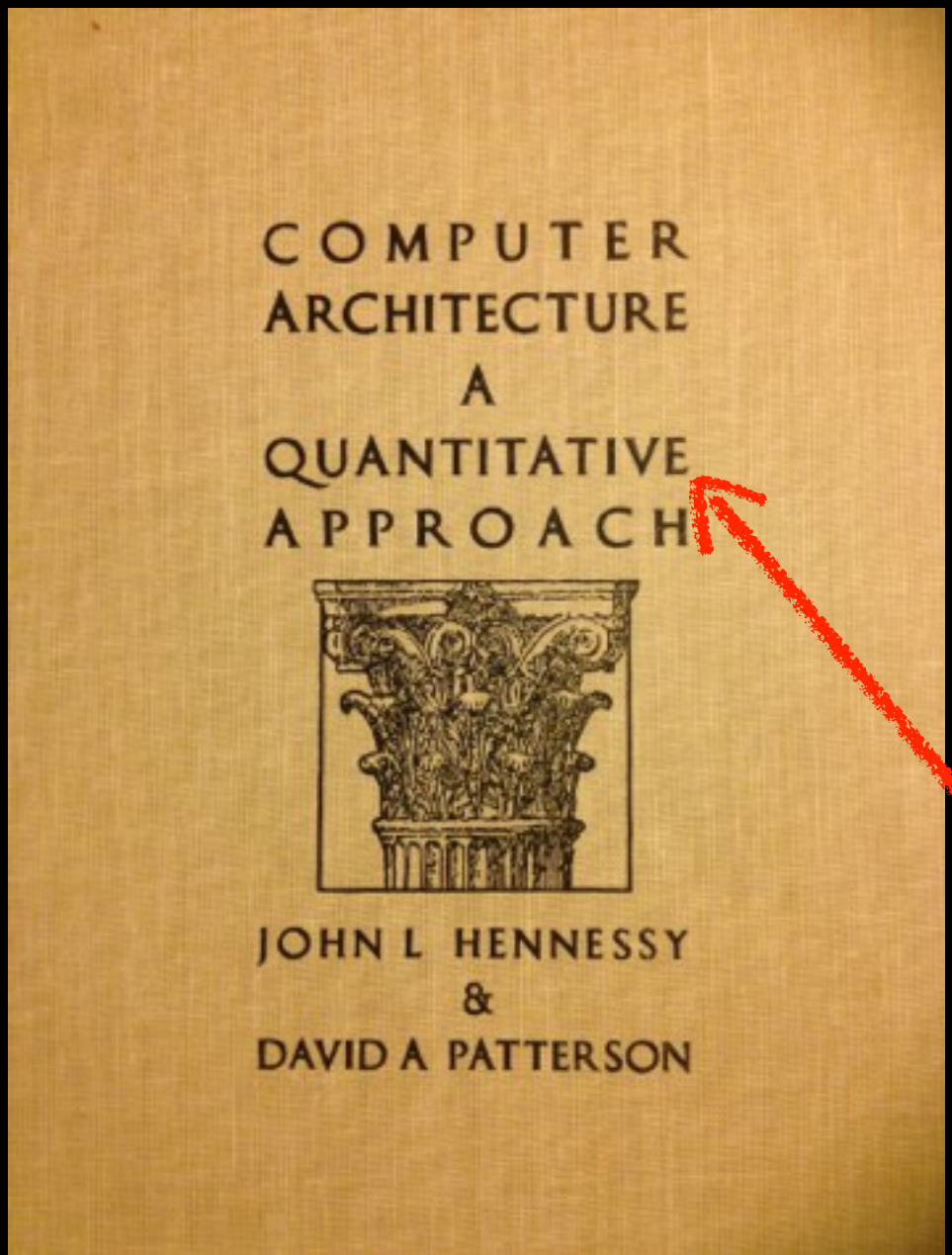
Play:

Today: Topics related to ISA design ...

- **MIPS-64: Highlights of Appendix A discussion of the RISC approach.**
- **Motorola 68000: Late 70s technology limitations led to a microcoded CPU.**
- **Short Break.**
- **Estimating the cost of a CPU chip.**

Instruction Set Design

Instruction Set Design



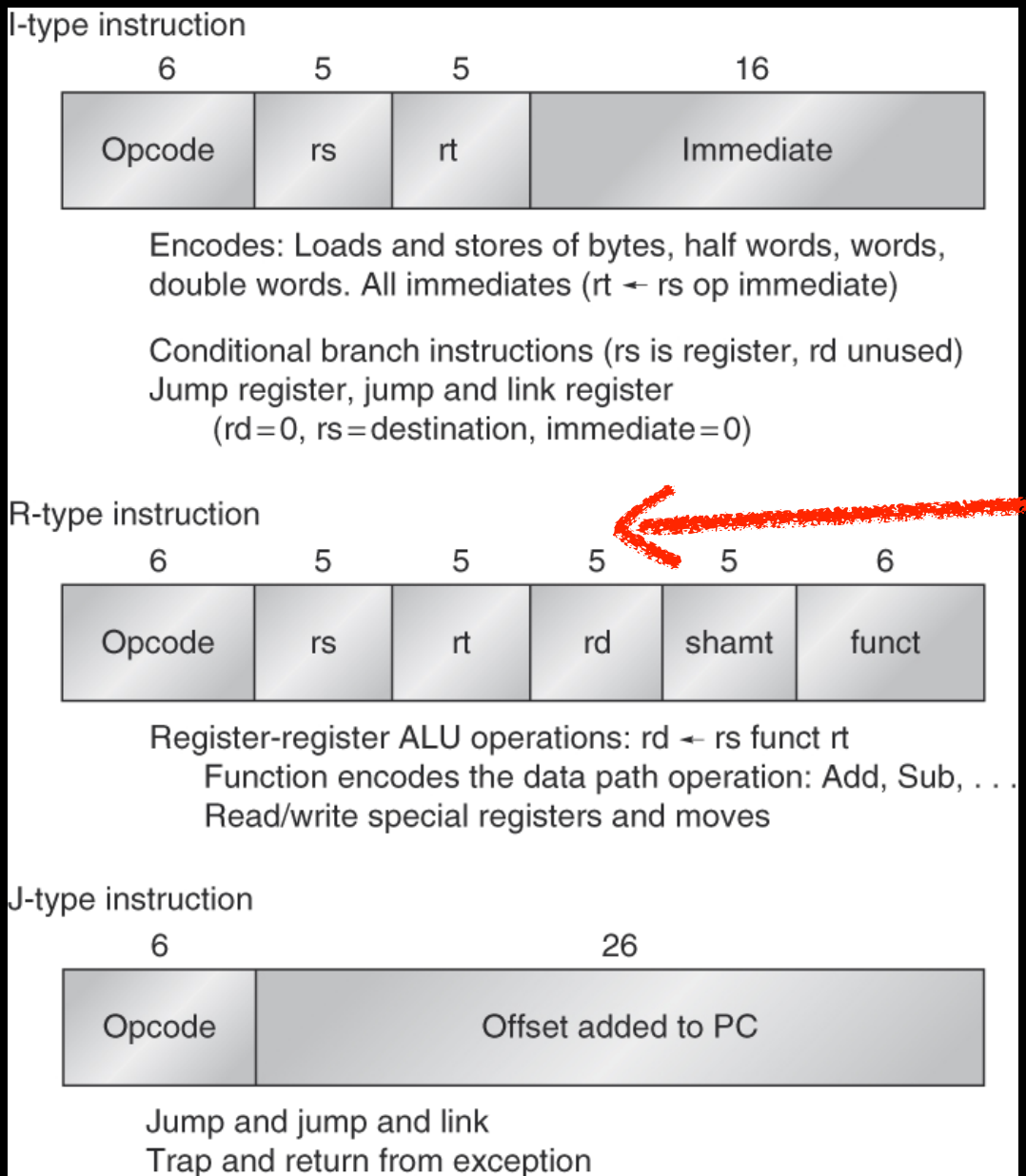
First edition, 1990

When defining a new instruction set, an architect makes about a dozen major decisions.

Qualitative reasoning, experience, and intuition will always play a major role in this effort ...

To what degree can these decisions be driven by quantitative data?

Examples from MIPS64



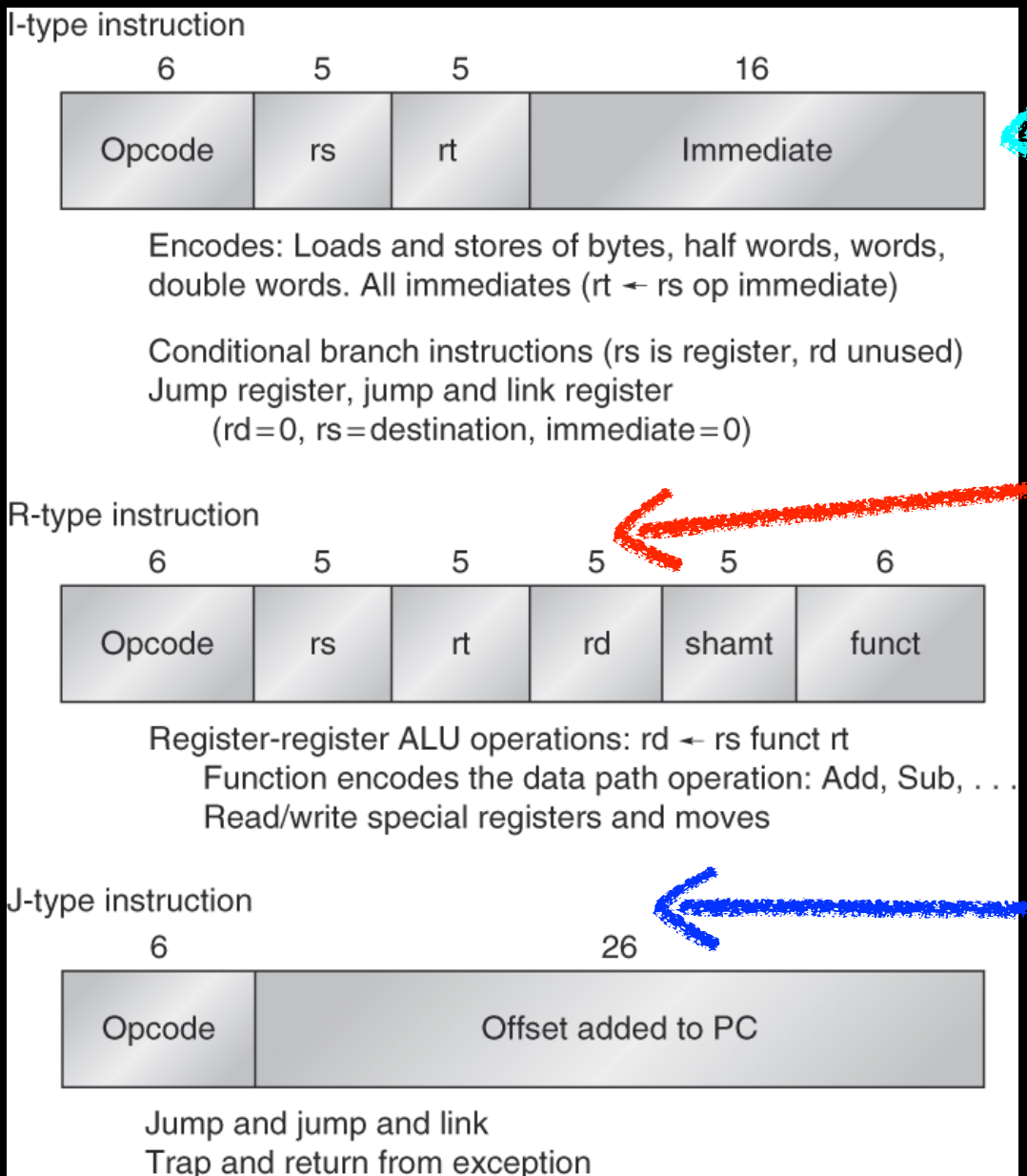
5th edition, Appendix A

Why use a 16-bit immediate field? Why not 12-bit? Or 20-bit?

Why use a 5-bit register field? Why is 32 general purpose registers the "right" number?

These questions are "quantitive" ... but the toolkit is more general.

Quantitative answers for qualitative questions …



Why so few load/store addressing modes?

Why no support of memory operands for ALU instructions?

Why restrict ISA to one fixed (32-bit) instruction size?

Appendix A systematically addresses these issues …

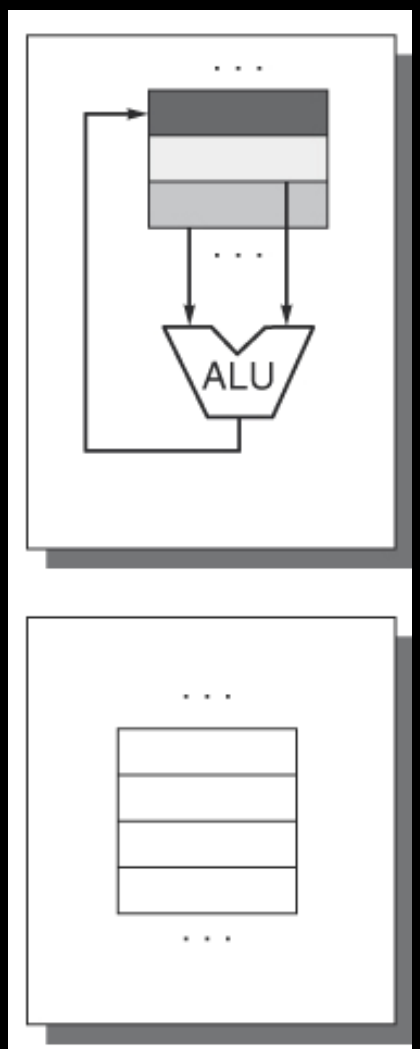
Register-register 1990s technology was ready for RISC

Type	Advantages	Disadvantages
Register-register (0, 3)	Simple, fixed-length instruction encoding. Simple code generation model. Instructions take similar numbers of clocks to execute (see Appendix C).	Higher instruction count than architectures with memory references in instructions. More instructions and lower instruction density lead to larger programs.

Machine code for
c = a + b;

Register (load-store)
Load R1,A
Load R2,B
Add R3,R1,R2
Store R3,C

Not really quantitative ...



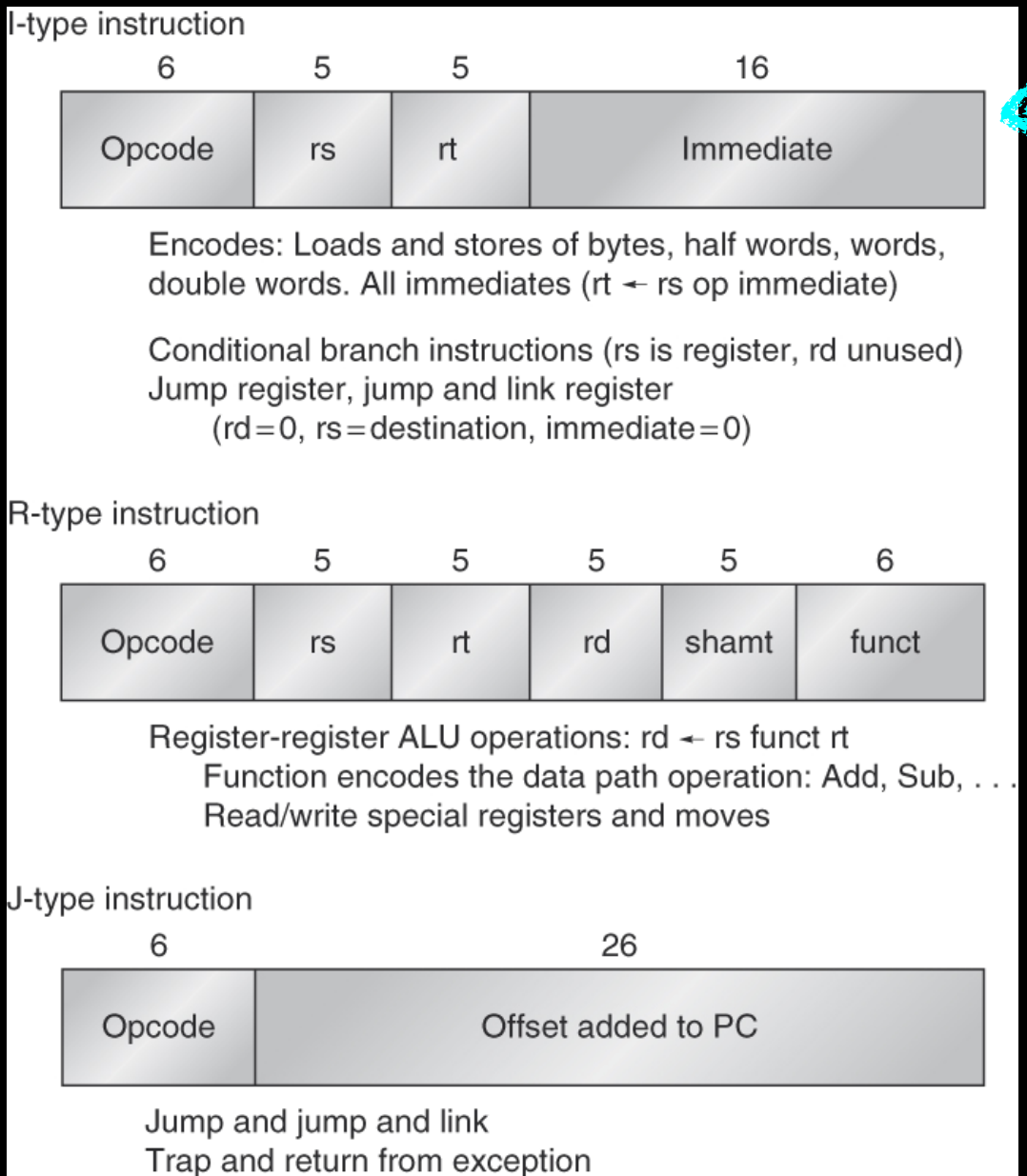
Transistors were available for on-chip instruction cache.

So, larger code size would not monopolize bandwidth to off-chip DRAM memory.

Fixed-length instructions made fast pipelining practical.

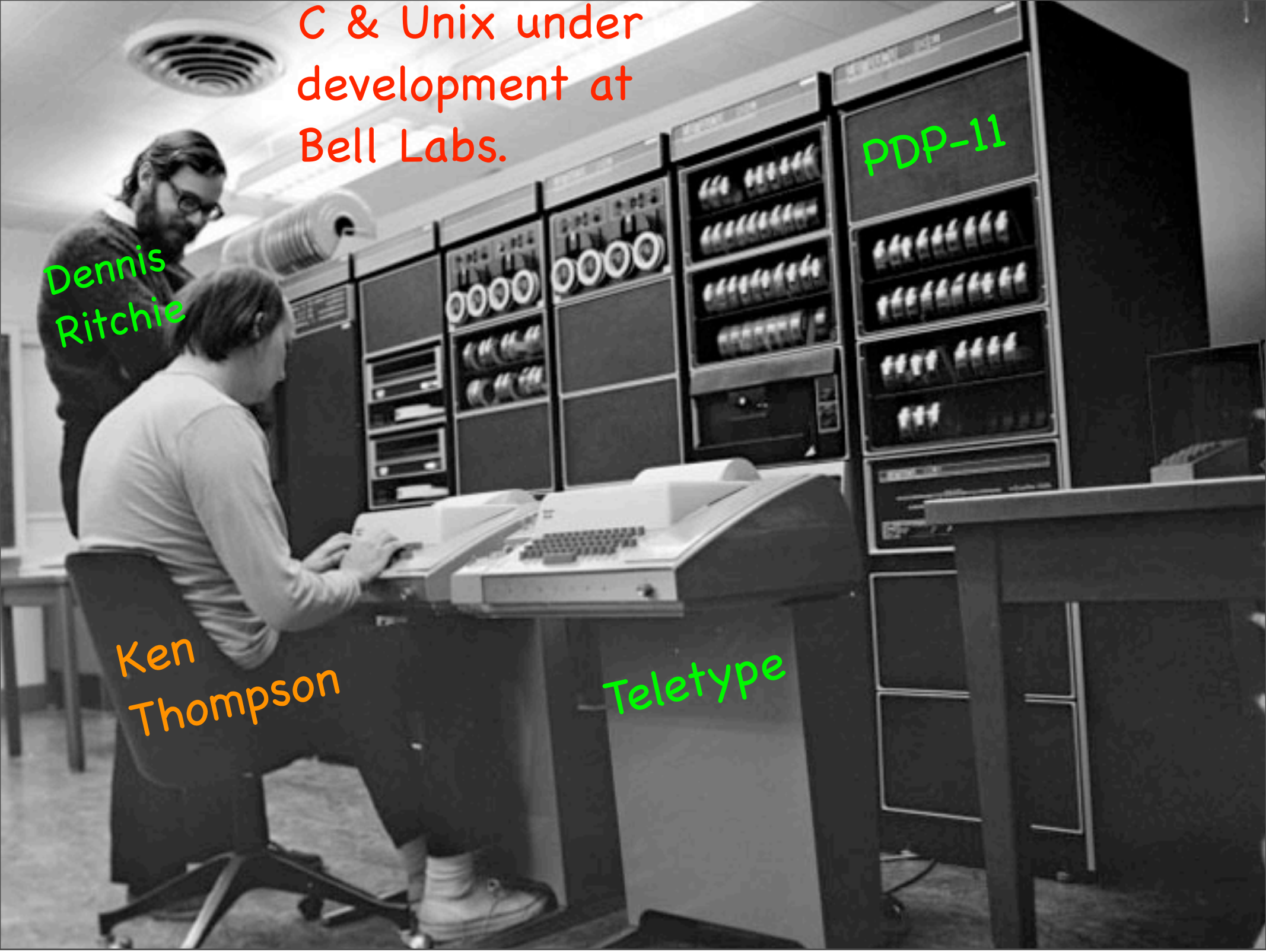
For the right target ISA, compiled code quality could match hand-coded assembly.

Quantitative answers for qualitative questions …



Why so few load/store addressing modes?

A quantitative approach: Measure dynamic instruction use of popular programs on machines with many instruction modes and a good compiler (VAX).


C & Unix under development at Bell Labs.
Dennis Ritchie
PDP-11
Ken Thompson
Teletype

Instruction modes: "C is a high-level assembler" origin

Addressing mode	Example instruction	Meaning	When used
Register	Add R4,R3	Regs[R4] ← Regs[R4] + Regs[R3]	When a value is in a register.
Immediate	Add R4,#3	Regs[R4] ← Regs[R4] + 3	For constants.
Displacement	Add R4,100(R1)	Regs[R4] ← Regs[R4] + Mem[100 + Regs[R1]]	Accessing local variables (+ simulates register indirect, direct addressing modes).
Register indirect	Add R4,(R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]	Accessing using a pointer or a computed address.
Indexed	Add R3,(R1 + R2)	Regs[R3] ← Regs[R3] + Mem[Regs[R1] + Regs[R2]]	Sometimes useful in array addressing: R1 = base of array; R2 = index amount.
Direct or absolute	Add R1,(1001)	Regs[R1] ← Regs[R1] + Mem[1001]	Sometimes useful for accessing static data; address constant may need to be large.
Memory indirect	Add R1,@(R3)	Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]	If R3 is the address of a pointer p, then mode yields $*p$.
Autoincrement	Add R1,(R2)+	Regs[R1] ← Regs[R1] + Mem[Regs[R2]] Regs[R2] ← Regs[R2] + d	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, d.
Autodecrement	Add R1, -(R2)	Regs[R2] ← Regs[R2] − d Regs[R1] ← Regs[R1] + Mem[Regs[R2]]	Same use as autoincrement. Autodecrement/-increment can also act as push/pop to implement a stack.
Scaled	Add R1,100(R2)[R3]	Regs[R1] ← Regs[R1] + Mem[100 + Regs[R2] + Regs[R3] * d]	Used to index arrays. May be applied to any indexed addressing mode in some computers.

w += i

w += 3

w += a[100 + i]

w += a[i]

w += a[i + j]

w += a[1001]

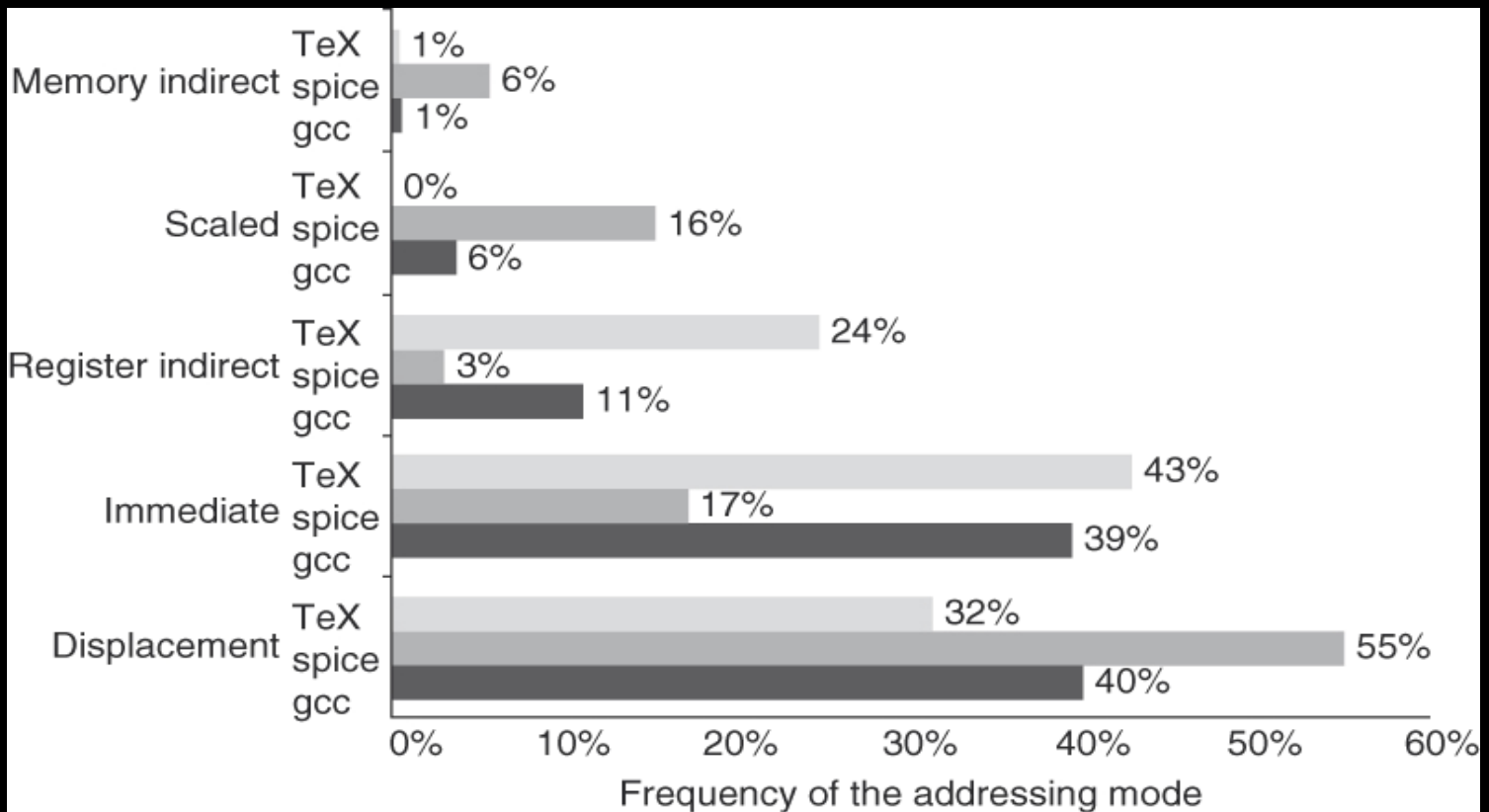
w += a[*p]

a[i++]

a[i--]

w += a[100 + i + d*j]

TeX, Spice, and Gcc, measured on a VAX

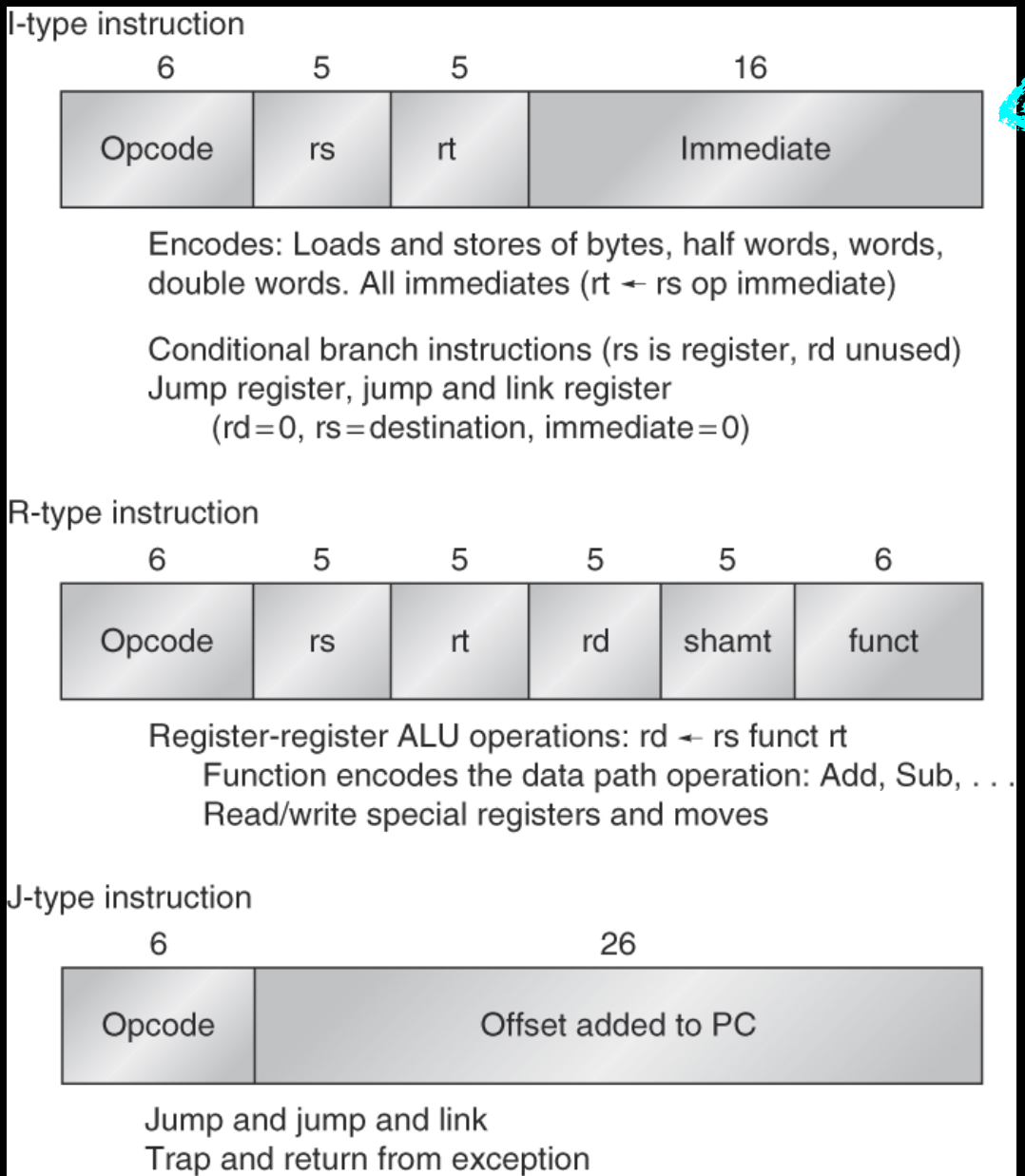


Quantitative answer: Register, displacement, and immediate are probably sufficient.

Register mode is 50%, and is not shown on chart.
So, divide percentages on chart by a factor of 2.

Addressing mode	Example instruction	Meaning	When used
Register	`Add R4,R3`	`Regs[R4] ← Regs[R4] + Regs[R3]`	When a value is in a register.
Immediate	`Add R4,#3`	`Regs[R4] ← Regs[R4] + 3`	For constants.
Displacement	`Add R4,100(R1)`	`Regs[R4] ← Regs[R4] + Mem[100 + Regs[R1]]`	Accessing local variables (+ simulates register indirect, direct addressing modes).
Register indirect	`Add R4,(R1)`	`Regs[R4] ← Regs[R4] + Mem[Regs[R1]]`	Accessing using a pointer or a computed address.

Quantitative answers for quantitative questions ...



Why use a 16-bit immediate field? Why not 12-bit? Or 20-bit?

A quantitative approach: Immediate field is used in several ways. Measure code traces for each, and pick a happy medium to keep instruction set simple.

ALU immediate usage (special case: load immediate)

Approach: Measure the size of the constant coded in the 32-bit VAX ALU immediate field, over a set of programs, by examining the dynamic instruction stream.

Data:

50% of the constants were <= 8 bits.

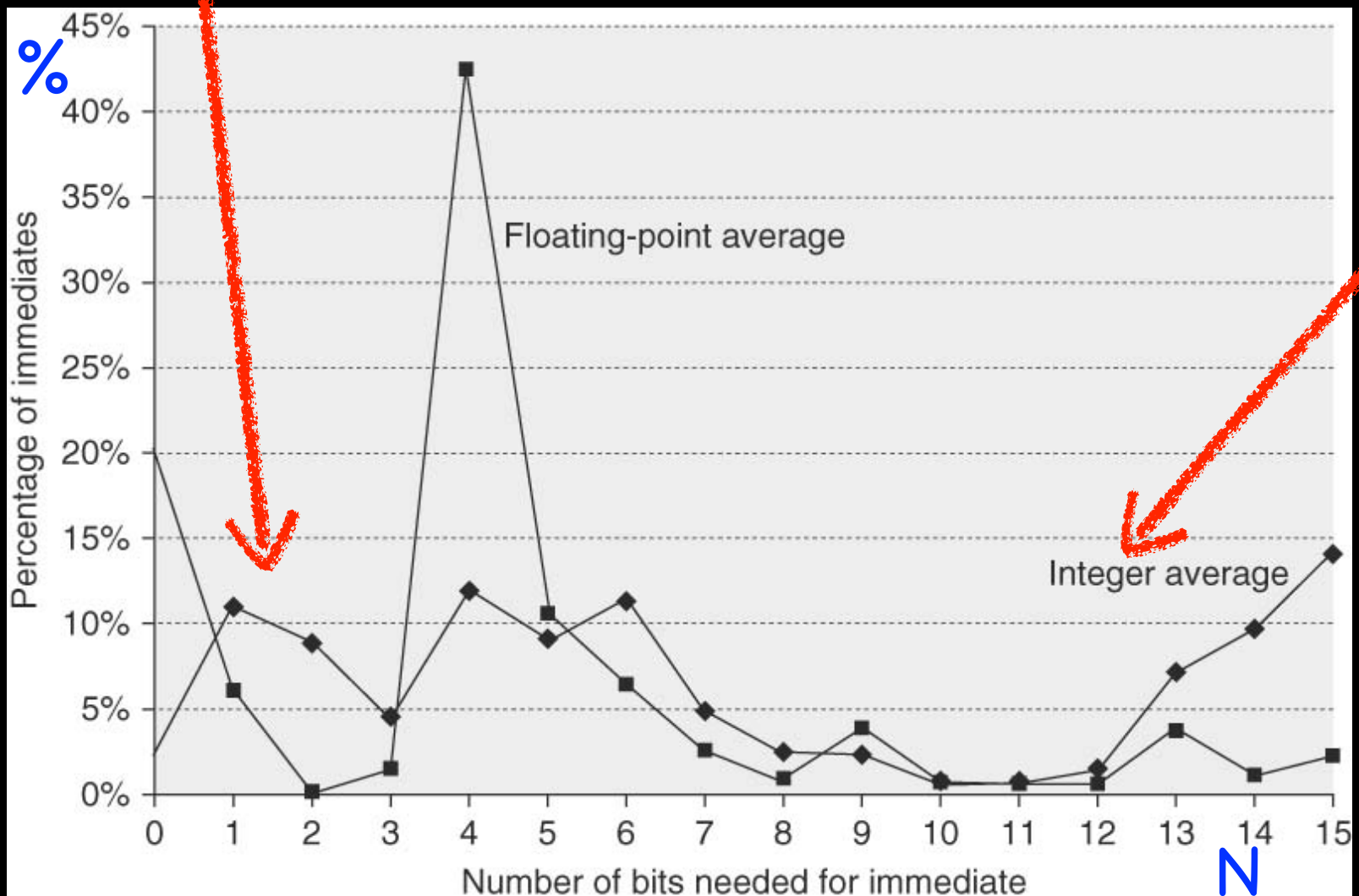
80% of the constants were <= 16 bits.

Conclusion:

16-bit ALU immediates are sufficient for the "common case".

DEC Alpha ISA: Has 16-bit ALU immediates

Small integers: popular for integer math.



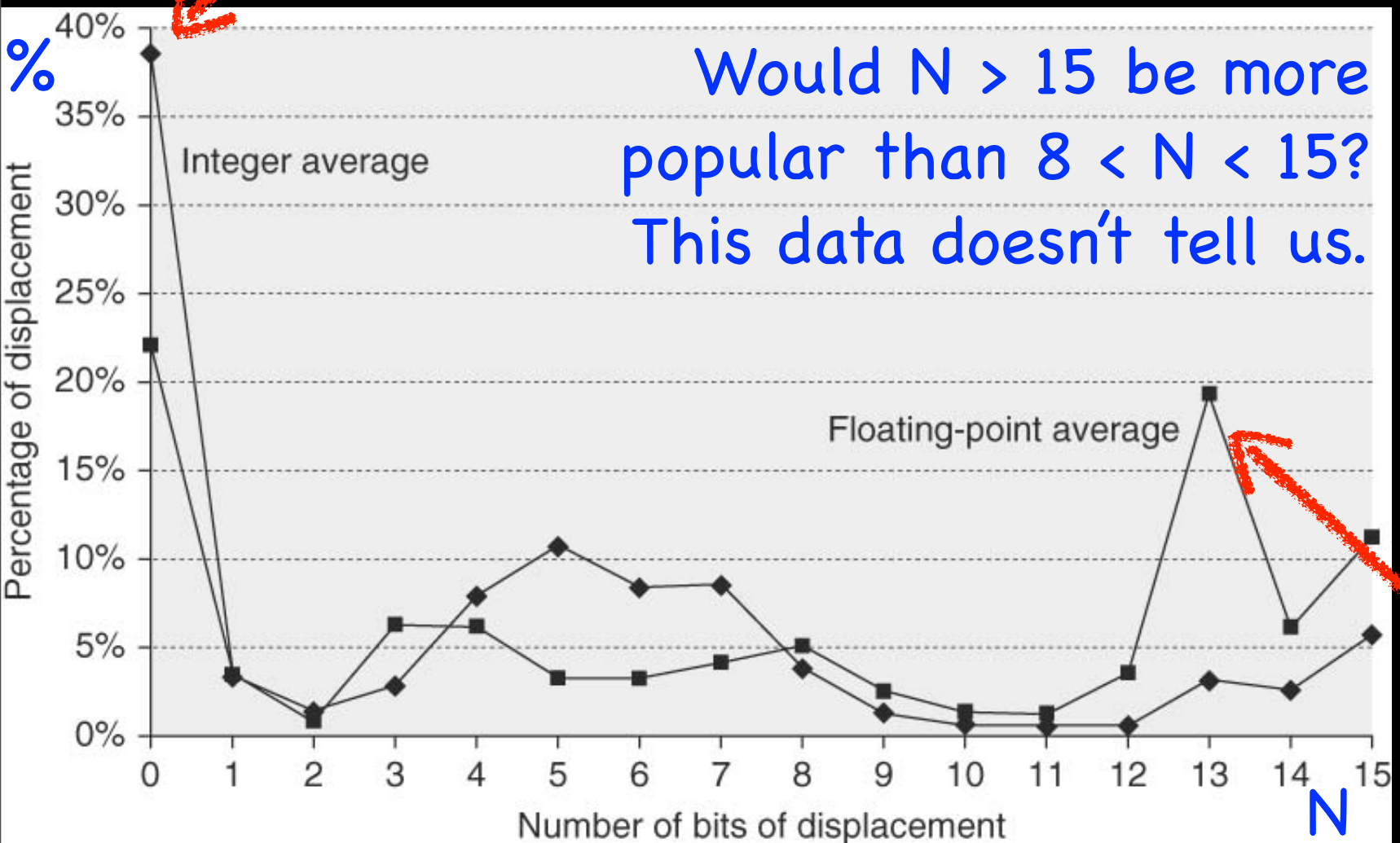
Large integers: Popular for address calculations.

Floats have unique properties.

The % of signed intermediates whose absolute values uses N out of 16 bits.

Dynamic plot for LW/SW displacement

Very small displacement are very popular.



Would N > 15 be more popular than 8 < N < 15? This data doesn't tell us.

The % of signed displacements whose absolute values uses N out of 16 bits.

Conclusions:

8 bits might be enough for integer common case. Need 16 bits for floats.

Popular sizes for vector and matrix blocks ..

And finally, conditional branch displacement

I-type instruction

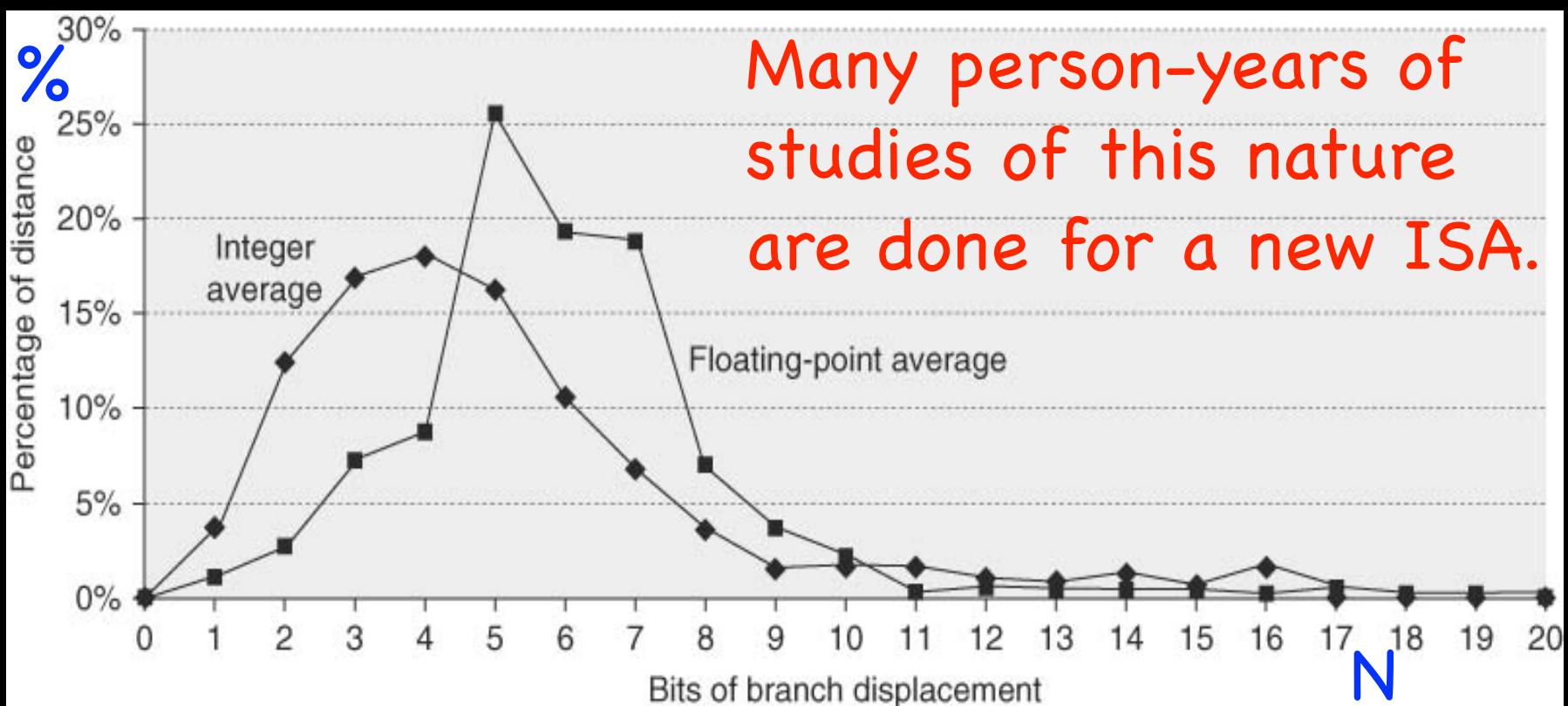
6	5	5	16
Opcode	rs	rt	Immediate

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)

16 bits will work well conditional branches, too.

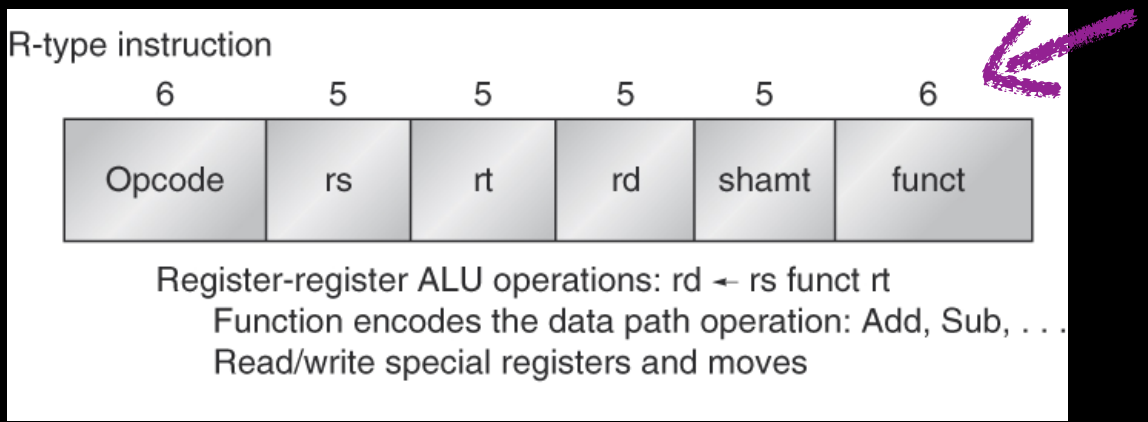
We have now justified all I-type immediate field uses quantitatively.



Many person-years of studies of this nature are done for a new ISA.

% of branch displacements whose abs() uses N out of 16 bits

How compiler technology can inform an ISA decision



Why use a 5-bit register field? Why is 32 general purpose registers the "right" number?

Setup:

Compilers will take high-level language arithmetic, and expand it through the use of "throwaway" temporary variables (temporaries written only once).

Example:

f = b + c + d - 1

becomes:

```
a := c + d
e := a + b
f := e - 1
```

Temporaries: a, e

How compiler technology can inform an ISA decision

Example:

f = b + c + d - 1

becomes:

```
a := c + d
e := a + b
f := e - 1
```

Temporaries: a, e

During code generation, a compiler allocates registers to temps when available, because registers are faster than memory.

```
a := c + d
e := a + b
f := e - 1
```

```
r1 := r2 + r3
r1 := r1 + r4
r1 := r1 - 1
```

In the general case, register allocation task is NP-complete ...

There are good heuristic solutions, but they require 16 free registers (preferably more) to work well.

This line of reasoning quantifies one advantage of 32 general purpose registers

Putting it all together: MIPS64 decisions

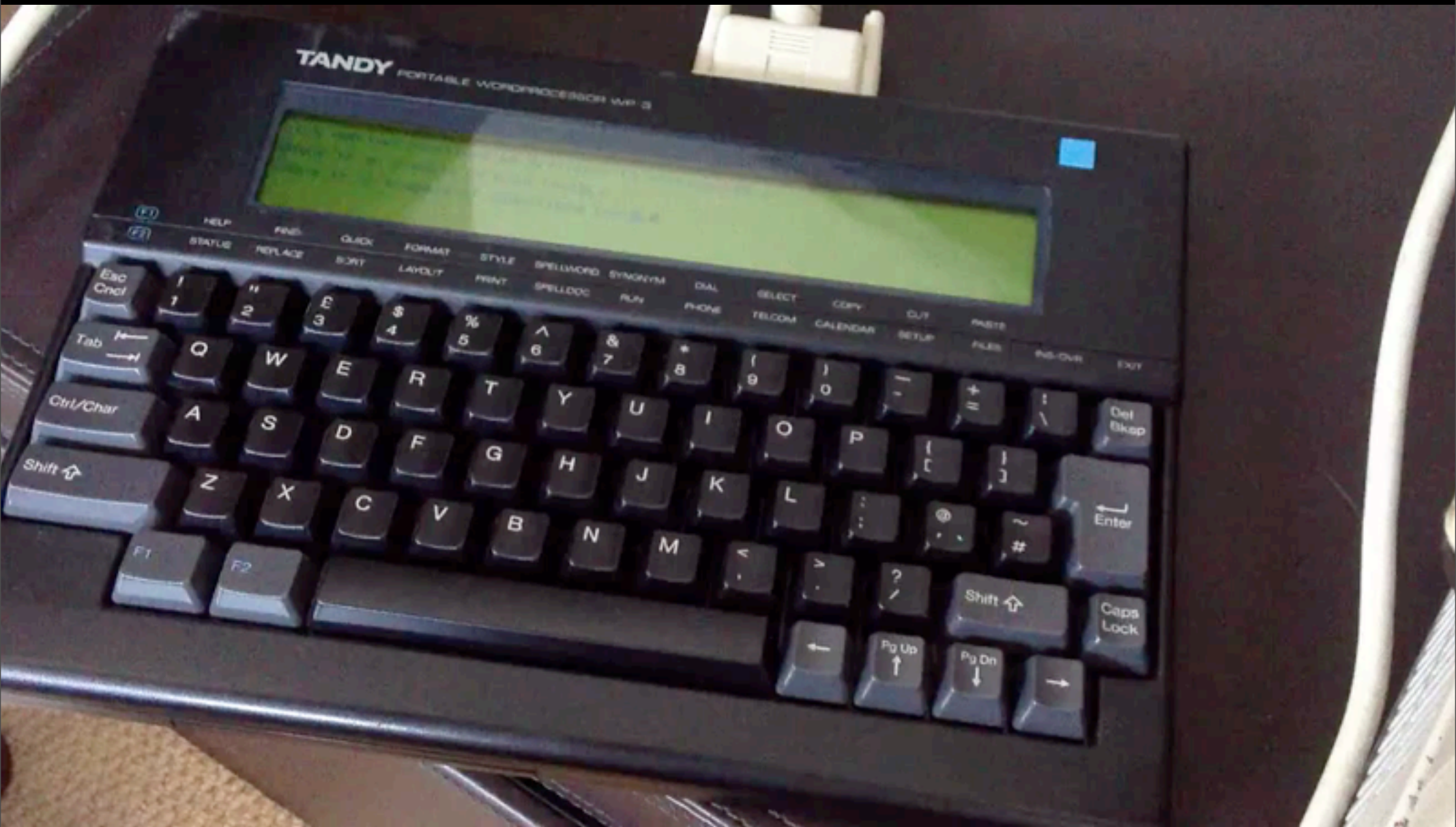
The take-away message of Appendix A is the methodology, not the particulars of the decisions.

- *Section A.2*—Use general-purpose registers with a load-store architecture.
- *Section A.3*—Support these addressing modes: displacement (with an address offset size of 12 to 16 bits), immediate (size 8 to 16 bits), and register indirect.
- *Section A.4*—Support these data sizes and types: 8-, 16-, 32-, and 64-bit integers and 64-bit IEEE 754 floating-point numbers.
- *Section A.5*—Support these simple instructions, since they will dominate the number of instructions executed: load, store, add, subtract, move register-register, and shift.
- *Section A.6*—Compare equal, compare not equal, compare less, branch (with a PC-relative address at least 8 bits long), jump, call, and return.
- *Section A.7*—Use fixed instruction encoding if interested in performance, and use variable instruction encoding if interested in code size.
- *Section A.8*—Provide at least 16 general-purpose registers, be sure all addressing modes apply to all data transfer instructions, and aim for a minimalist instruction set.

Last week: 30th anniversary of the original Macintosh

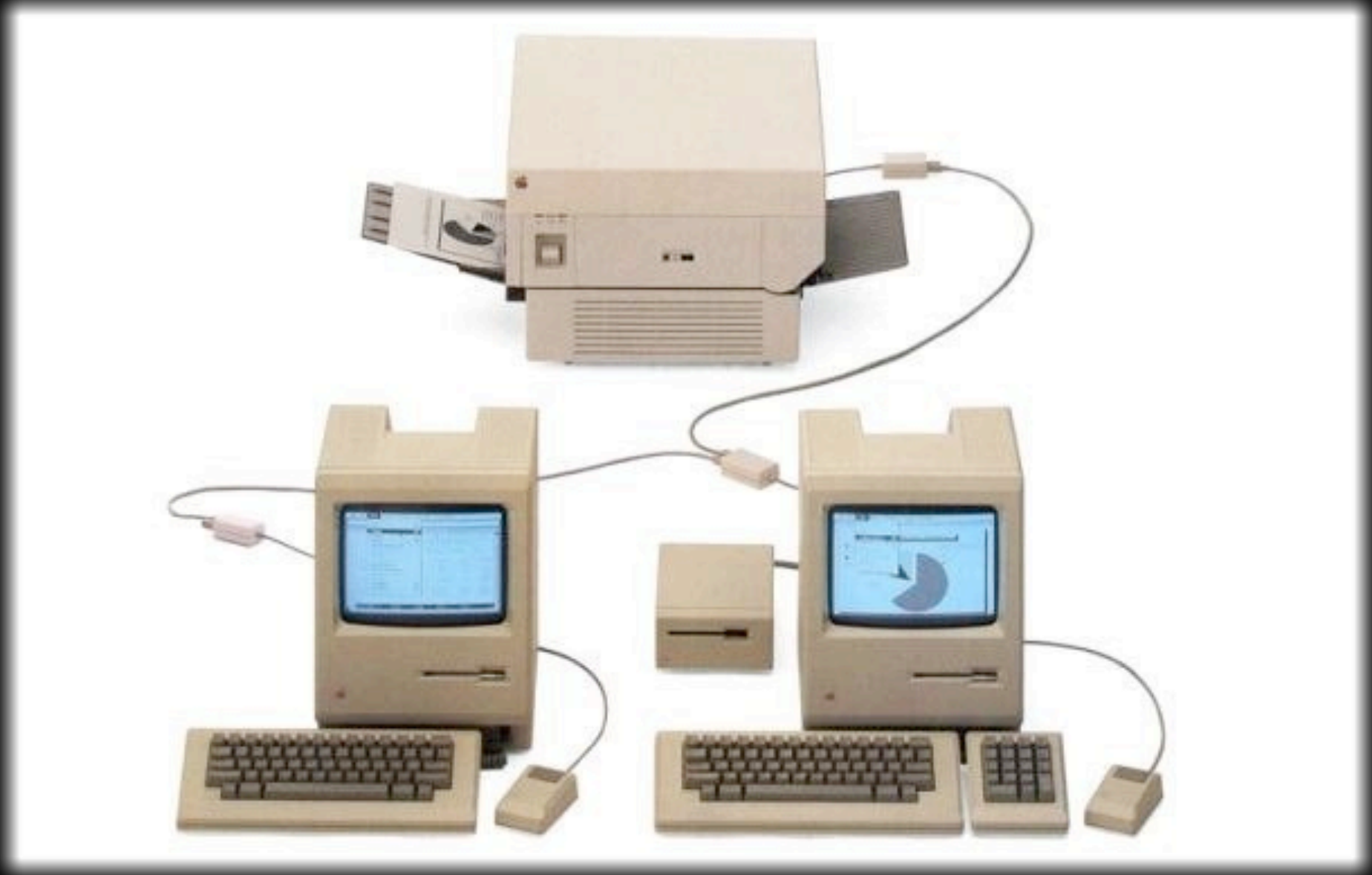
It brought desktop publishing out of the labs …

“Mechatronic” automatic typewriter as printer …



“Daisy-wheel” printer, invented by Diablo Inc.

Networked workstations sharing a quiet laser printer …



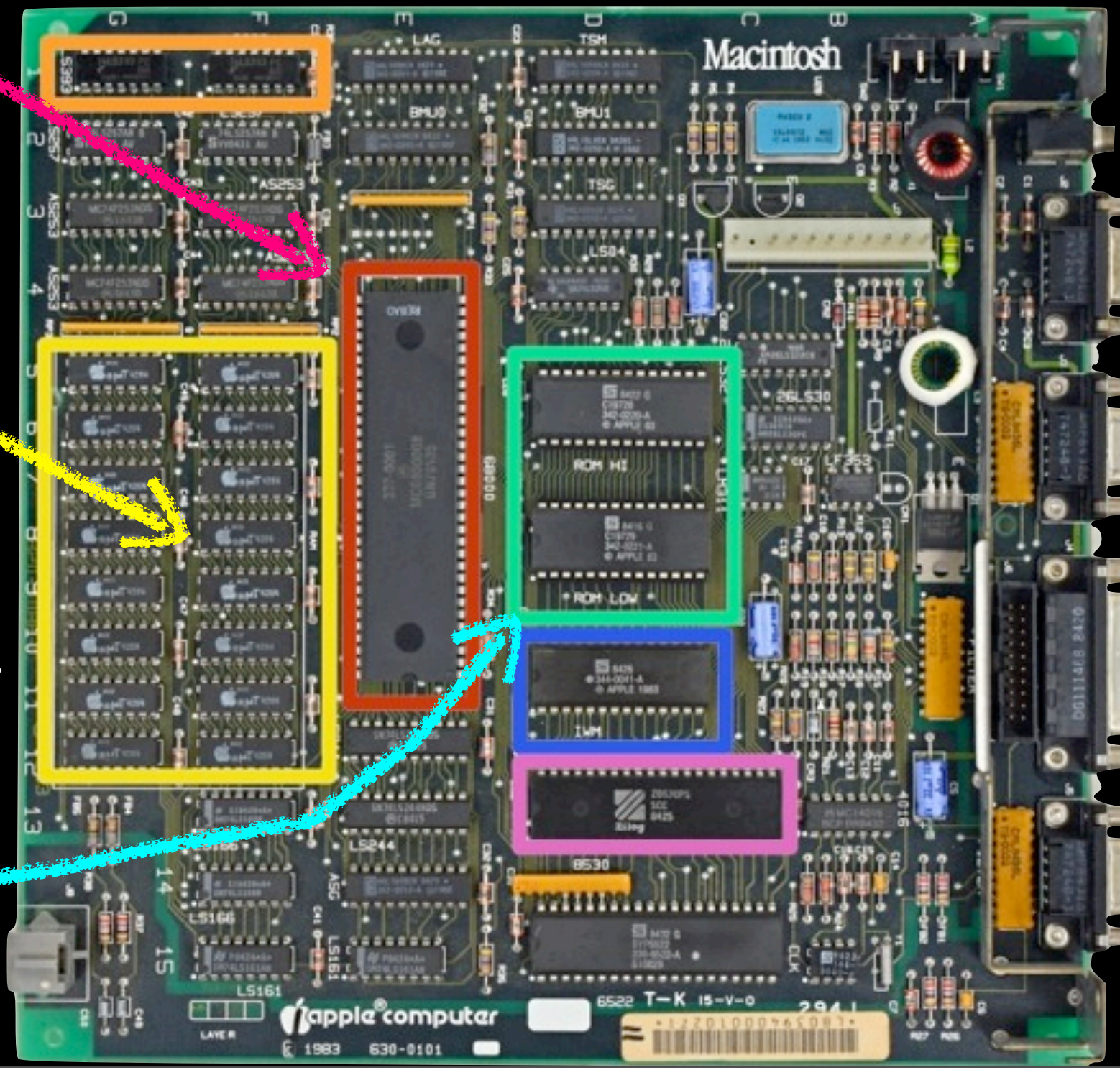
The reason why most people bought computers …
until the Internet reached critical mass (mid-1990s).

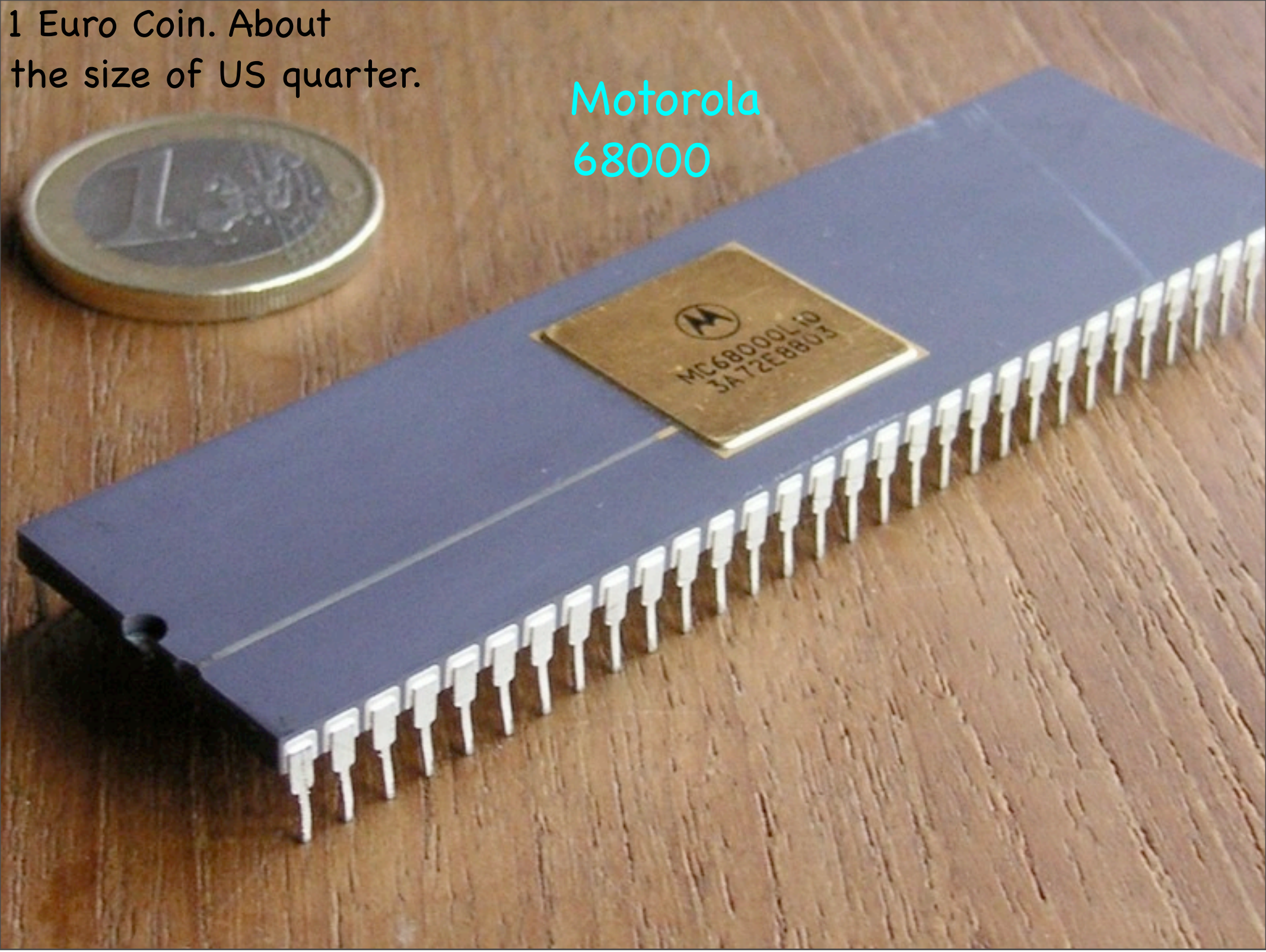
Mainboard of the original Macintosh

Motorola 68000 CPU. Control uses microcode.

128KB DRAM. 22KB was devoted to frame buffer for 512 x 342 1-bit video.

64KB ROM. Held OS, library code.




1 Euro Coin. About
the size of US quarter.
Motorola
68000
MC68000L10
3A72E8803

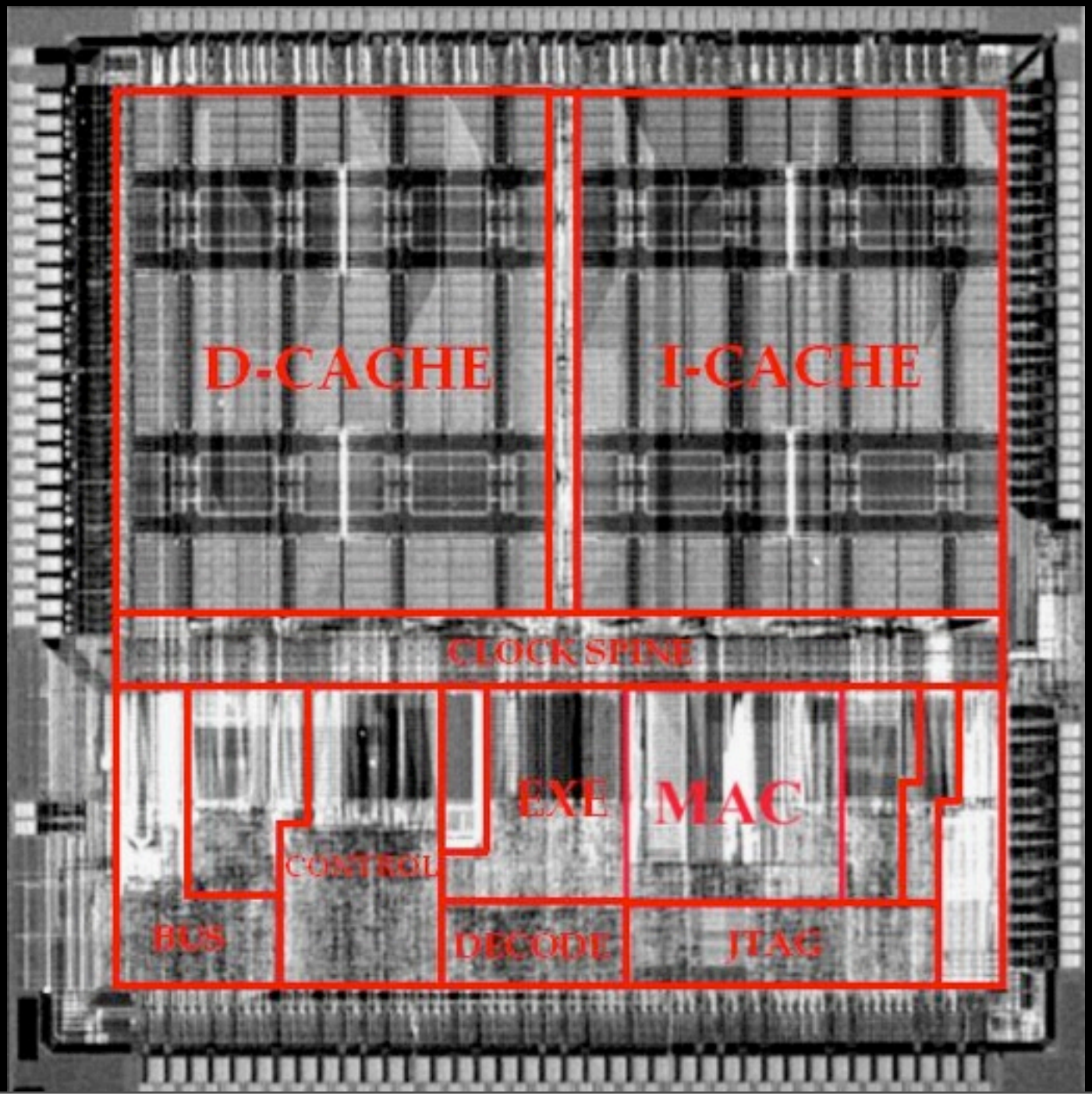
ARM CPU

By 2001, silicon/package technology made CPU instruction bandwidth a minor issue.

32 KB instruction cache uses 3 million transistors

Typical miss rate: 1.5%

DRAM interface uses 61 pins that toggle at 100 MHz



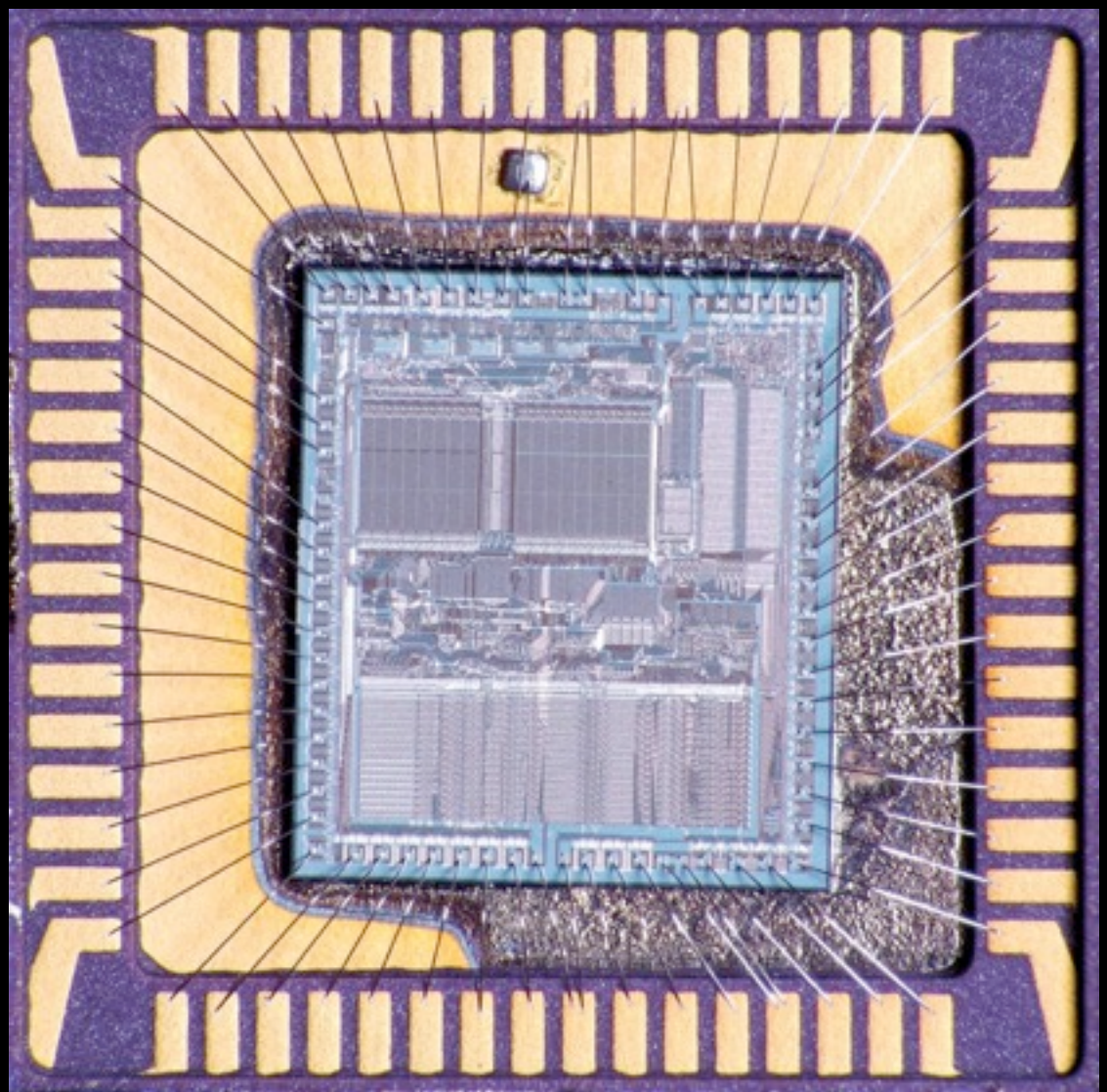
ARM CPU

32 KB instruction cache uses 3 million transistors

Typical miss rate: 1.5%

DRAM interface uses 61 pins that toggle at 100 MHz

CPU instruction bandwidth issues defined architecture.



Original Mac had 128K DRAM. Code size very important too.

Moto 68000

Released: 1979

68,000 transistors

No room for caches

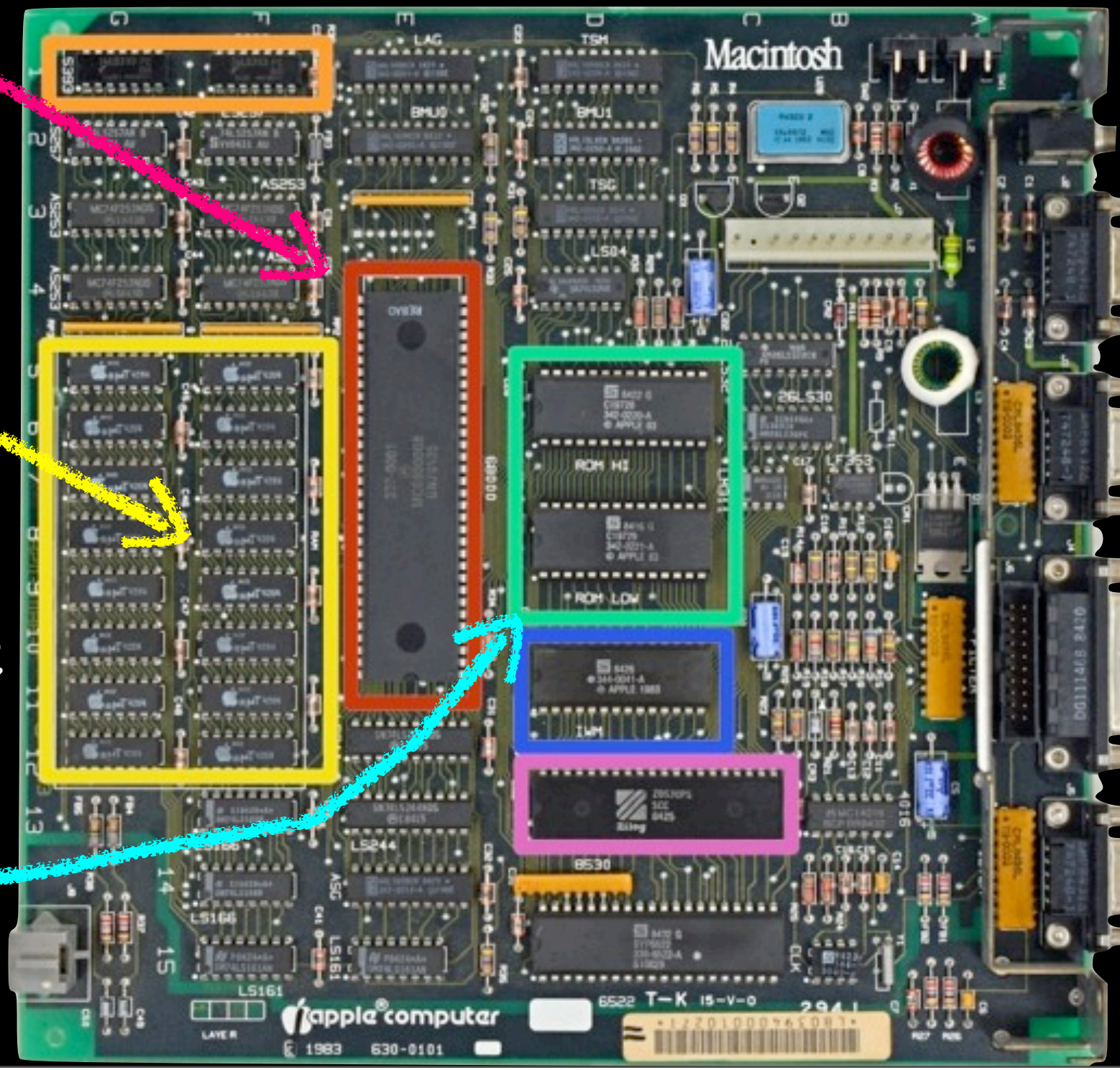
64-pin package. Multiplexed memory bus. 290 ns read (3.5 MHz) for 8 MHz CPU clock

Recall: Original Macintosh mainboard

Motorola 68000 CPU. Control uses microcode.

128KB DRAM. 22KB was devoted to frame buffer for 512 x 342 1-bit video.

64KB ROM. Held OS, library code.



Register-memory

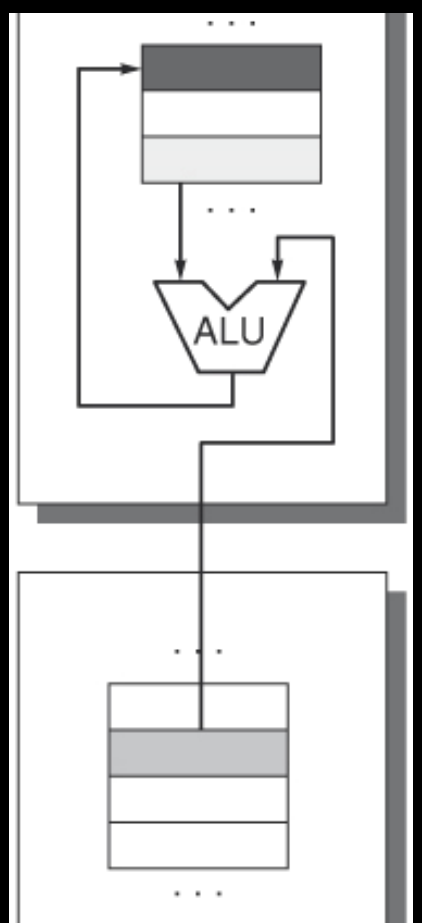
No i-cache? Slow memory bandwidth? Expensive DRAM? This might help ...

Type	Advantages	Disadvantages
Register-memory (1, 2)	Data can be accessed without a separate load instruction first. Instruction format tends to be easy to encode and yields good density.	Operands are not equivalent since a source operand in a binary operation is destroyed. Encoding a register number and a memory address in each instruction may restrict the number of registers. Clocks per instruction vary by operand location.

Machine code for c = a + b;

Register (register-memory)
Load R1,A
Add R3,R1,B
Store R3,C

Recall: 4 instrs for reg-reg.



Without cache or fast bus, slow instruction fetch is inevitable.

Register-memory lets us amortize each instruction fetch with a data fetch.

Complex instructions lets CPU do something useful while waiting for memory.

Performance equation for 1979 technologies …

$$\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \quad \frac{\text{Cycles}}{\text{Instruction}} \quad \frac{\text{Seconds}}{\text{Cycle}}$$

Goal is to optimize execution time, not individual equation terms.

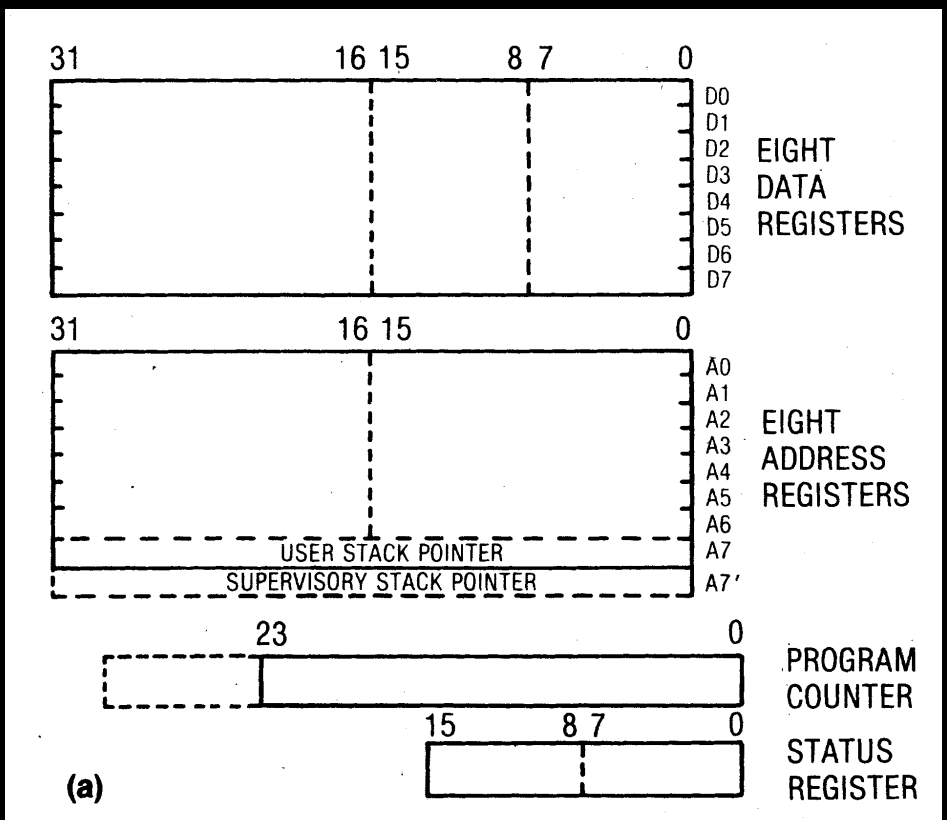
Define instructions that do a lot of work, so a program fetches fewer of them.

Memory speed sets minimum CPI. Complex instruction can do something useful during the wait.

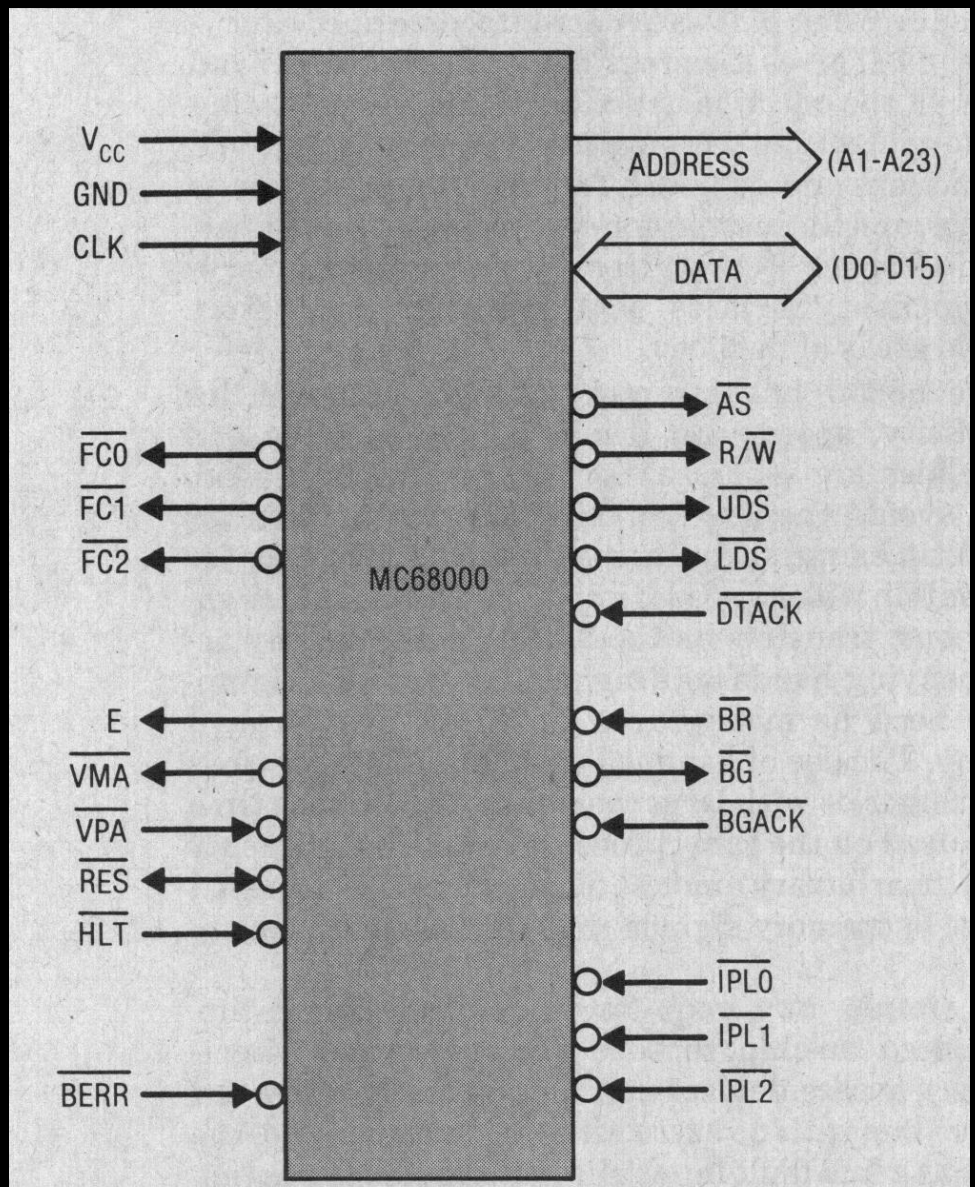
The more on-chip work our complex instructions let us do, the faster the clock should be.

68000 programmer's model

2 banks of 8 32-bit registers: one for arithmetic ("data"), and one for memory address calculations.

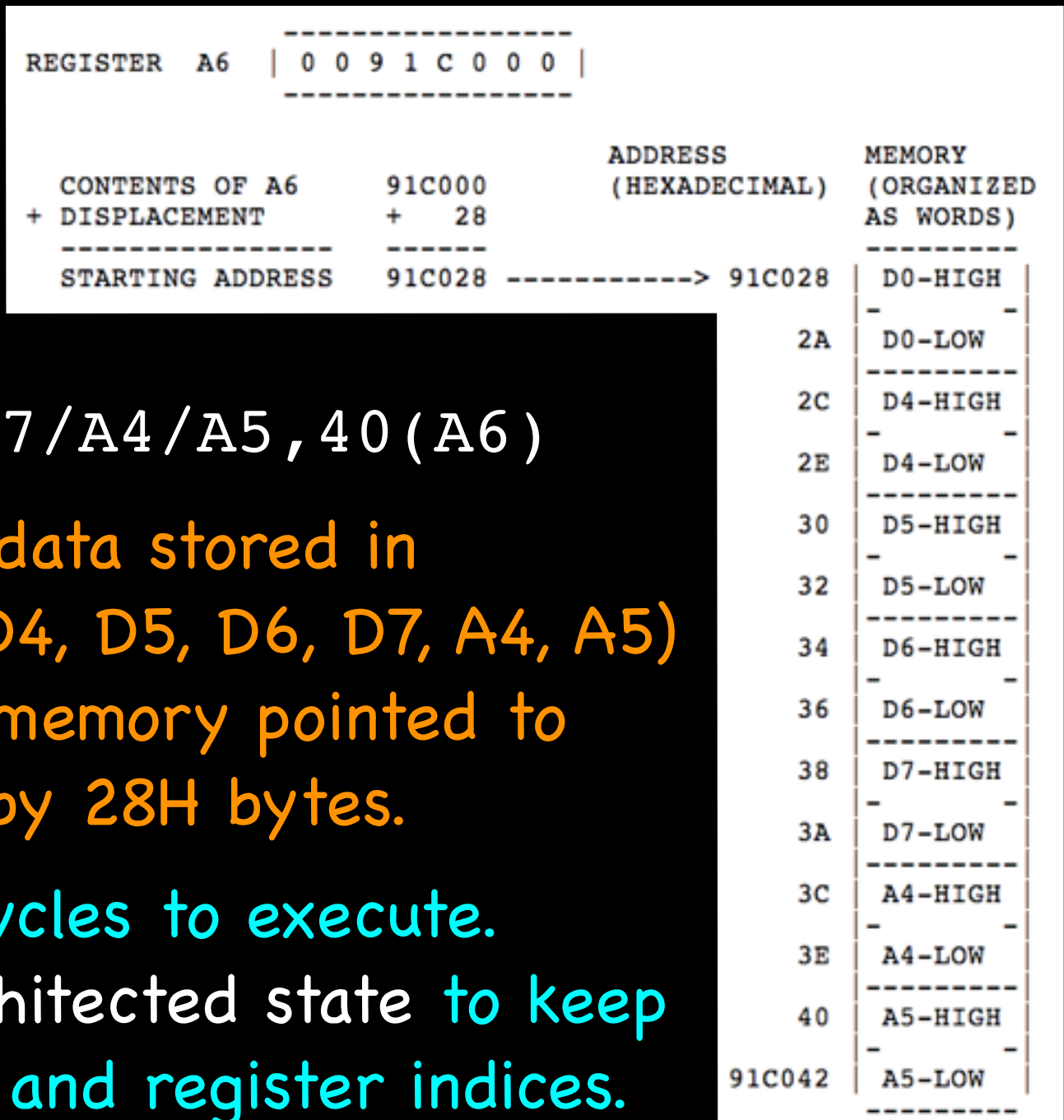


Linear address space.
24-bit bus -> 16 MB memory



An example of a complex instruction

8 byte instruction fetch amortized by 28 byte data move




```
MOVEM.L D0/D4-D7/A4/A5,40(A6)
```

Move the 32-bit data stored in 7 registers (D0, D4, D5, D6, D7, A4, A5) to the region of memory pointed to by A^, displaced by 28H bytes.

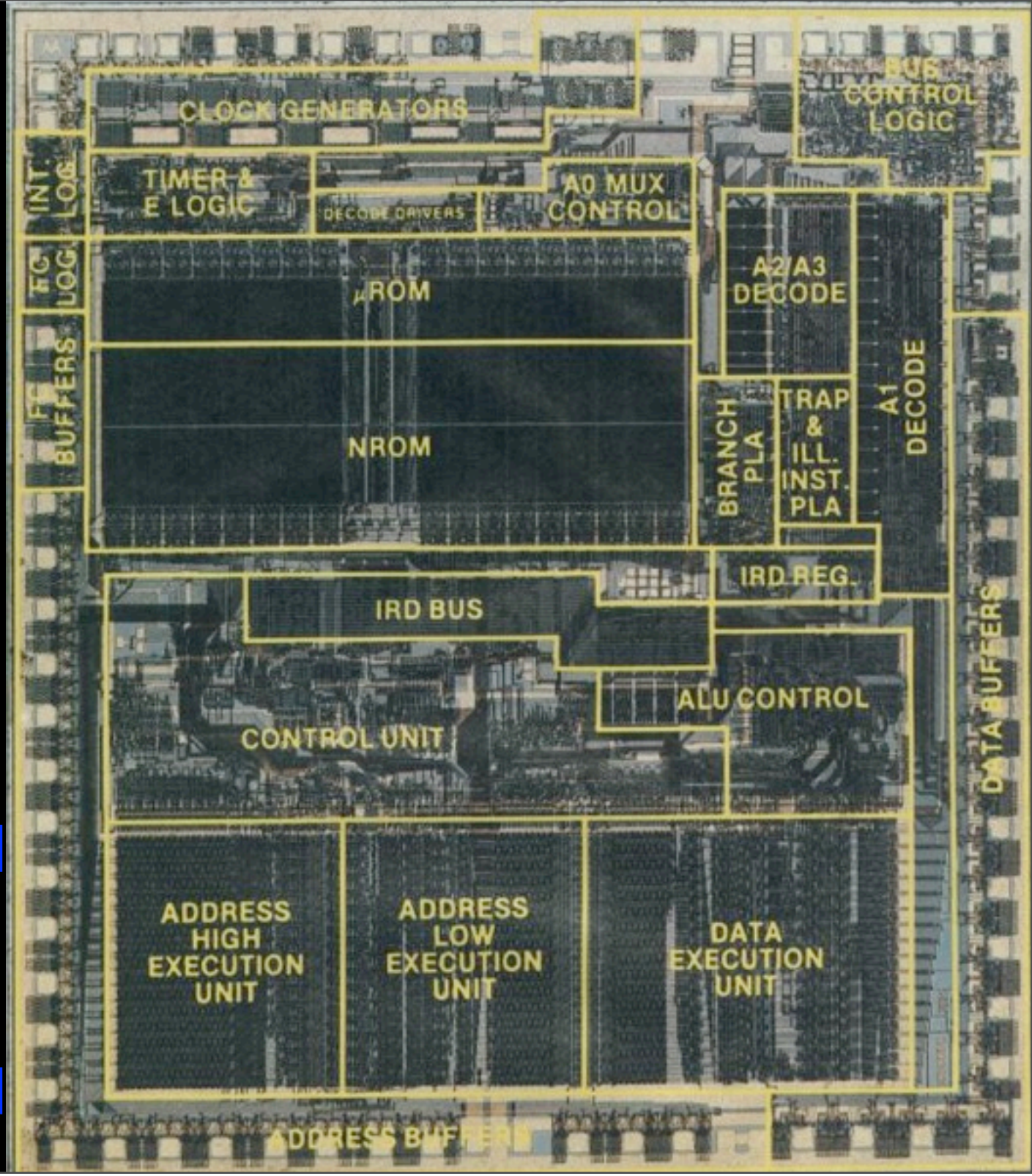
Takes 58 clock cycles to execute.

Requires non-architected state to keep track of memory and register indices.

How do we design a CPU that executes 58-cycle instructions?

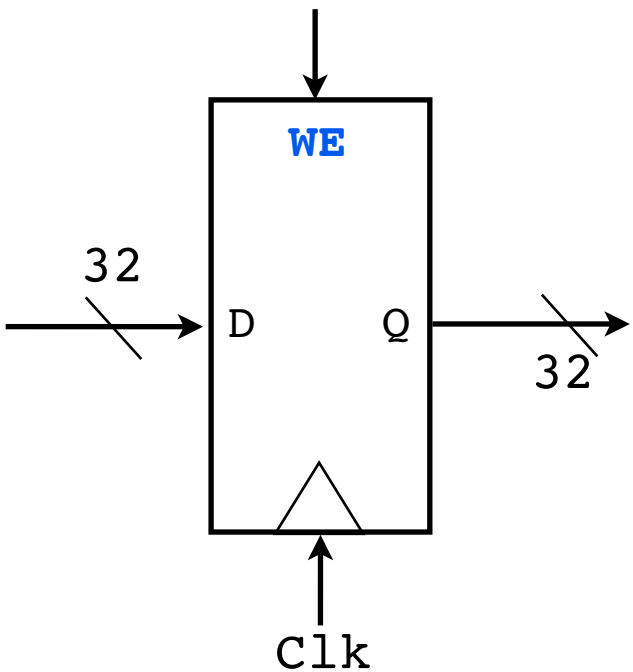
Microcode controller: Takes up most of chip area.

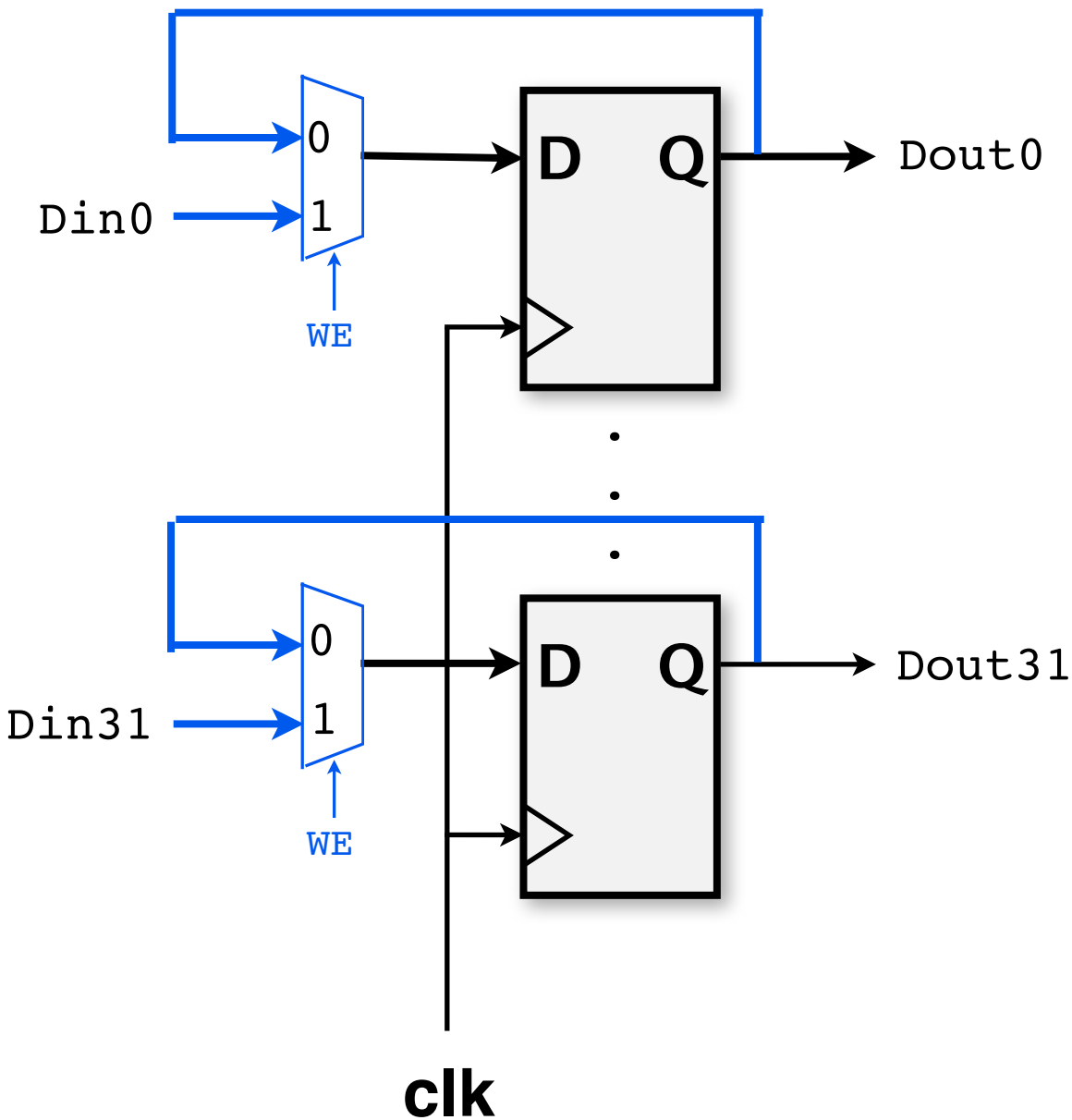
Datapath: Specialized for multi-cycle operation.



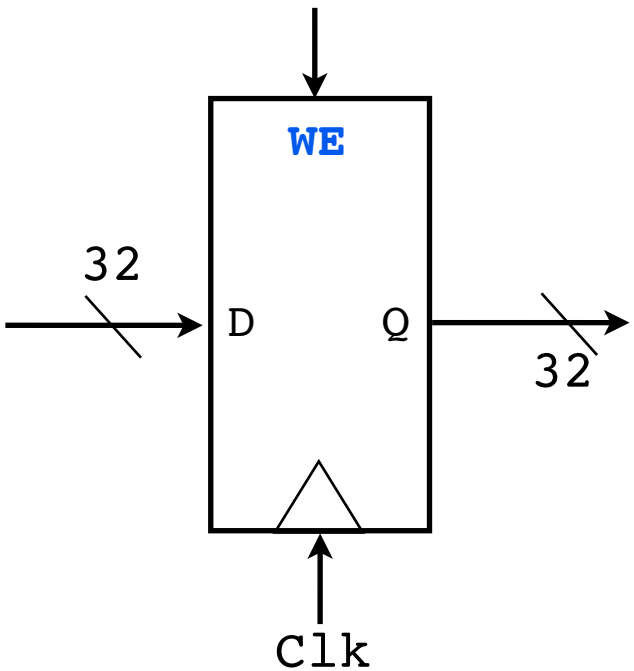
Preliminaries

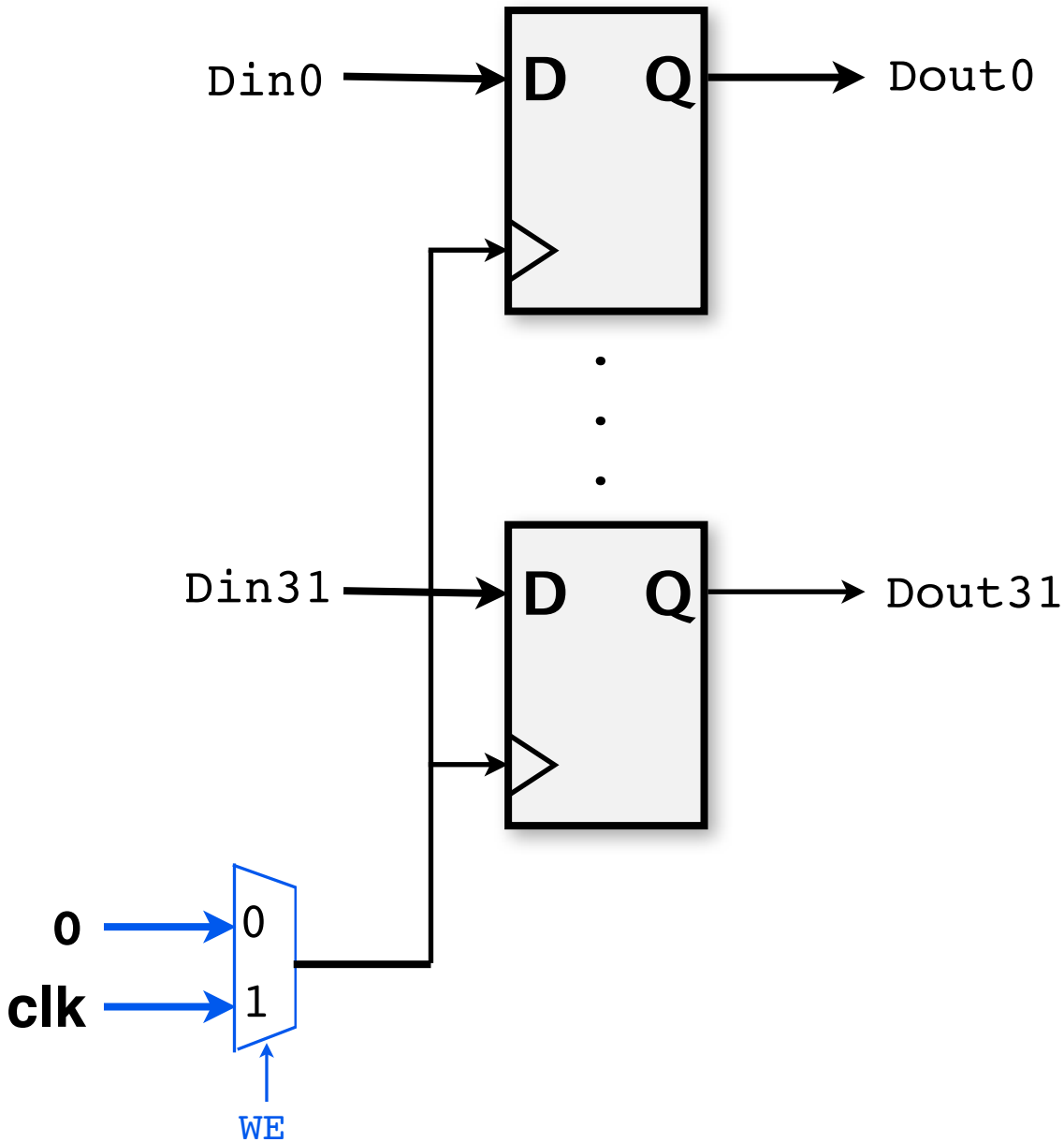
Revisiting register write enables ...



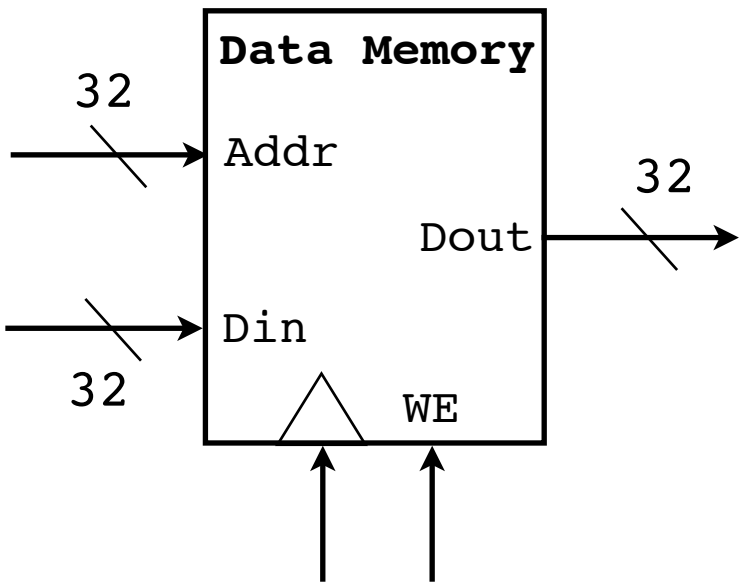


Another approach to write enables ...





What's inside our data memory?

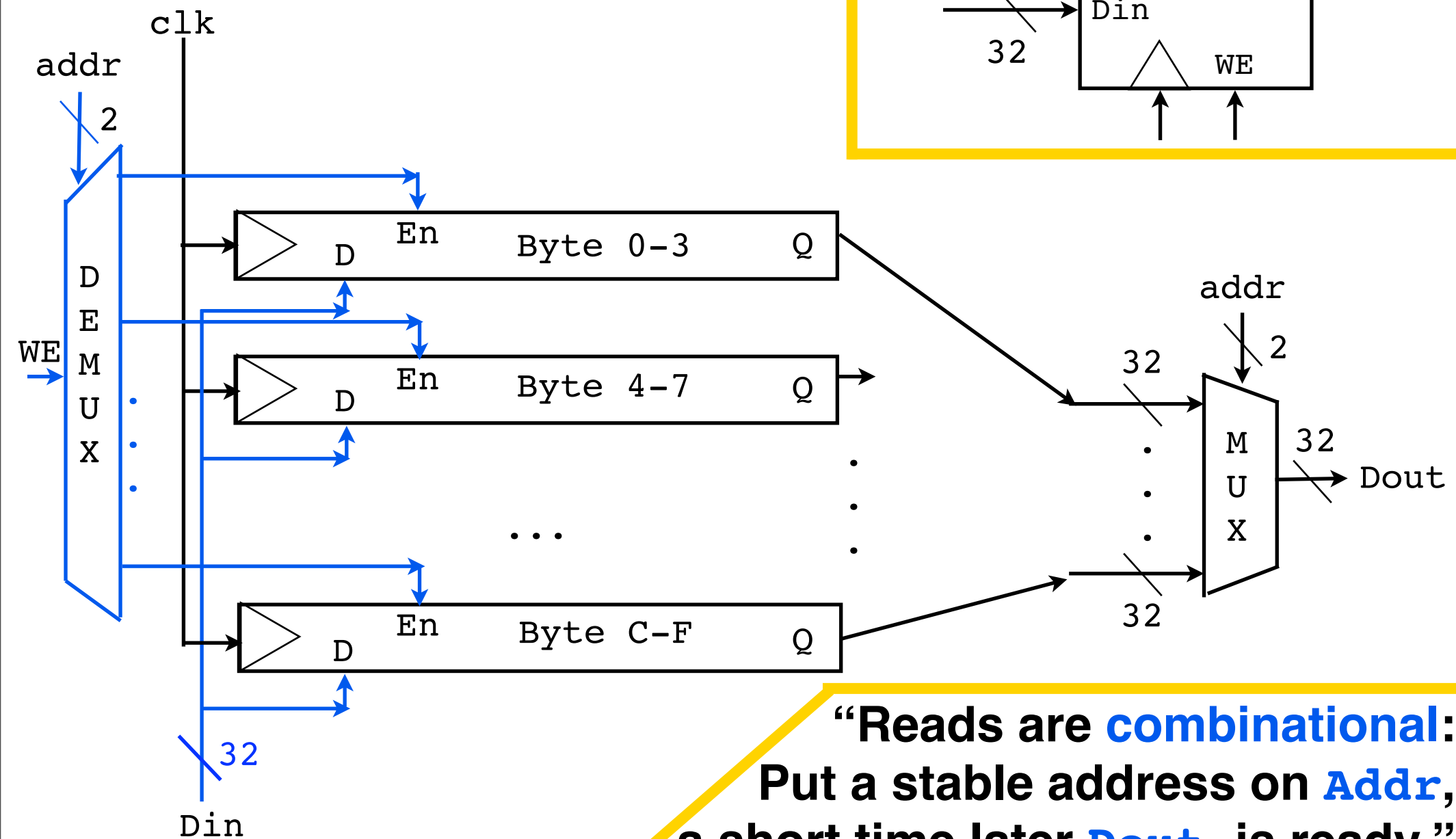


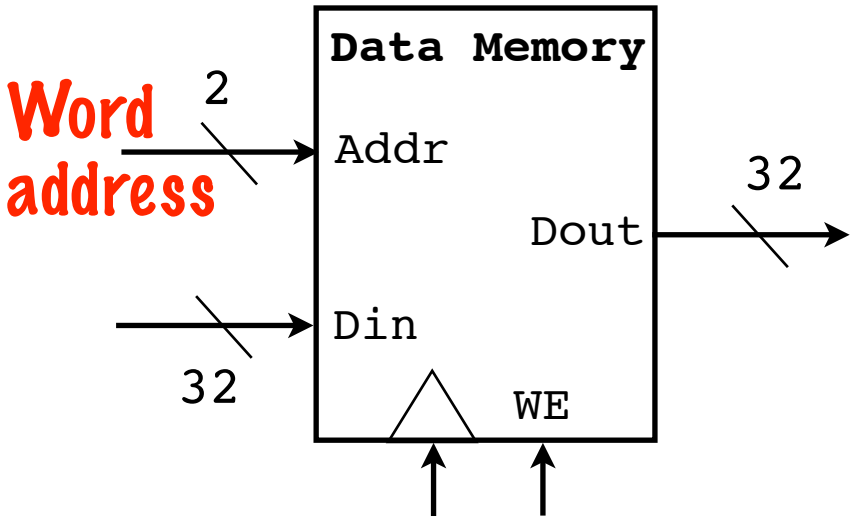
Reads are combinational:
Put a stable address on `Addr`,
a short time later `Dout` is ready.

Writes are clocked: If `WE` is high, memory `Addr` captures `Din` on positive edge of clock.

Let's modify our register file to make one ...

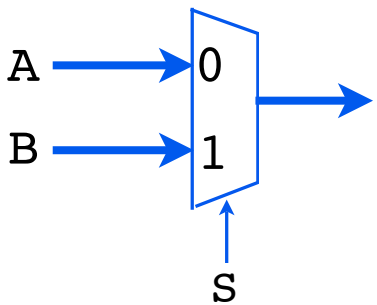
"Writes are clocked: If WE is high, memory Addr captures Din on positive edge of clock."

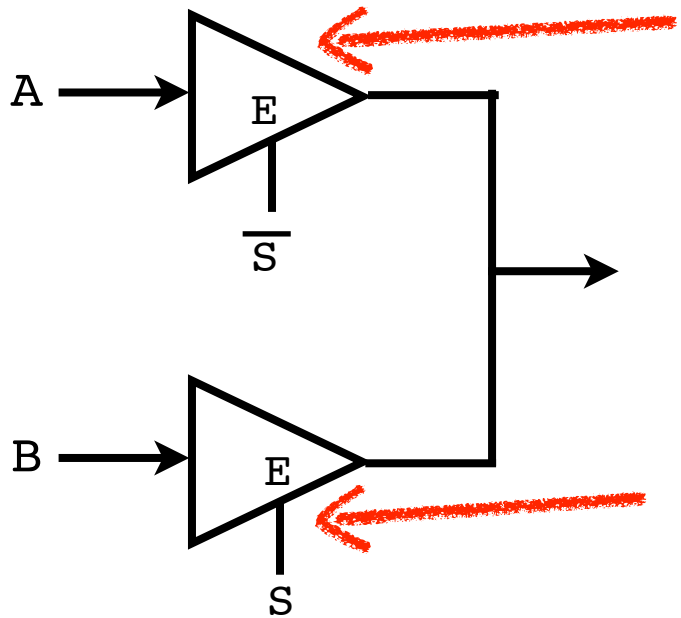




"Reads are combinational: Put a stable address on Addr, a short time later Dout is ready."

What's inside a multiplexer?





A tri-state buffer.

If E=1, it drives its input to its output.

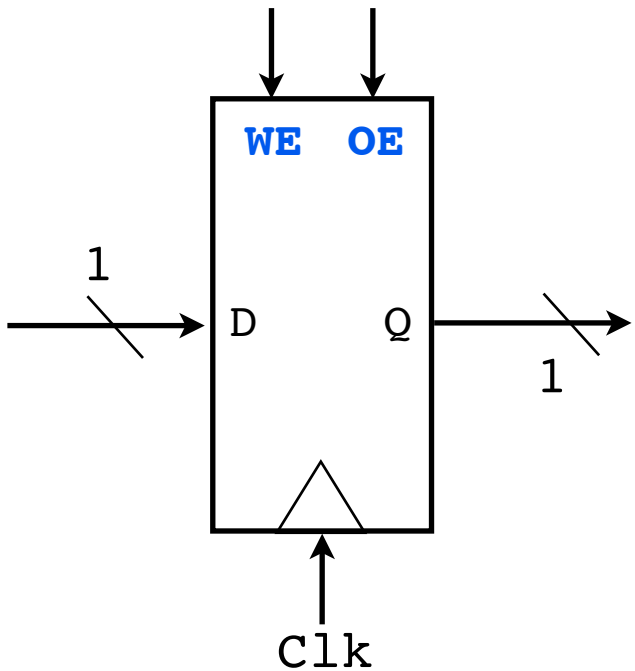
If E=0, it acts as though it was erased from the schematic.

This multiplexer design ensures exactly one buffer drives the output at any time.

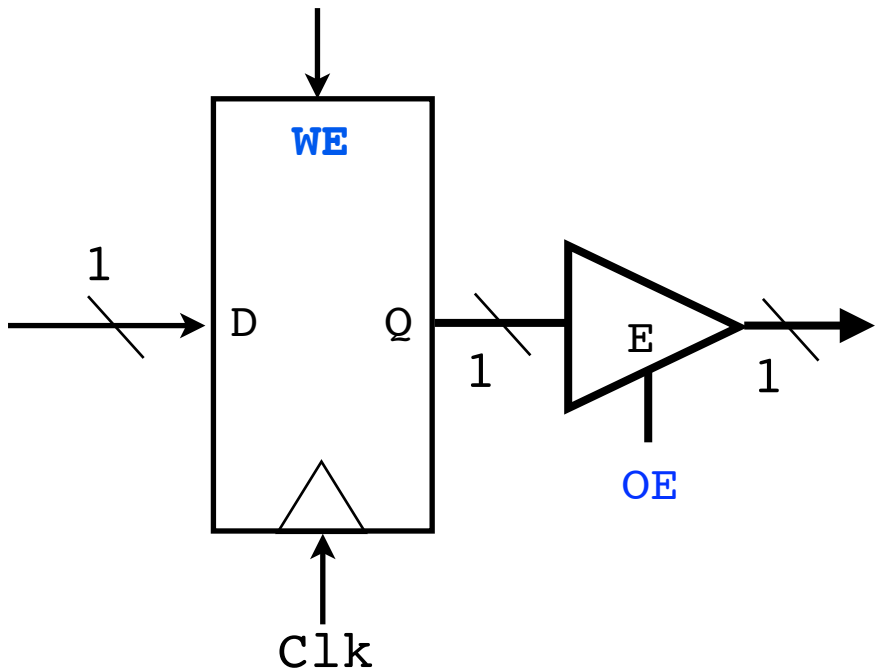
Why do we need to know this?

"Output enable" on registers

An output-enabled register.



How to design one ...



The CPU design we will show soon uses registers with WE and OE.

The designer has the responsibility to ensure exactly one output-enabled register is driving a shared wire at any time.

One part of the 68000 datapath

16 least-significant bits of the 8 data registers.
Each has 2 WEs and 2 OEs (one for each bus).

Two 16-bit buses

Access to off-chip memory bus.

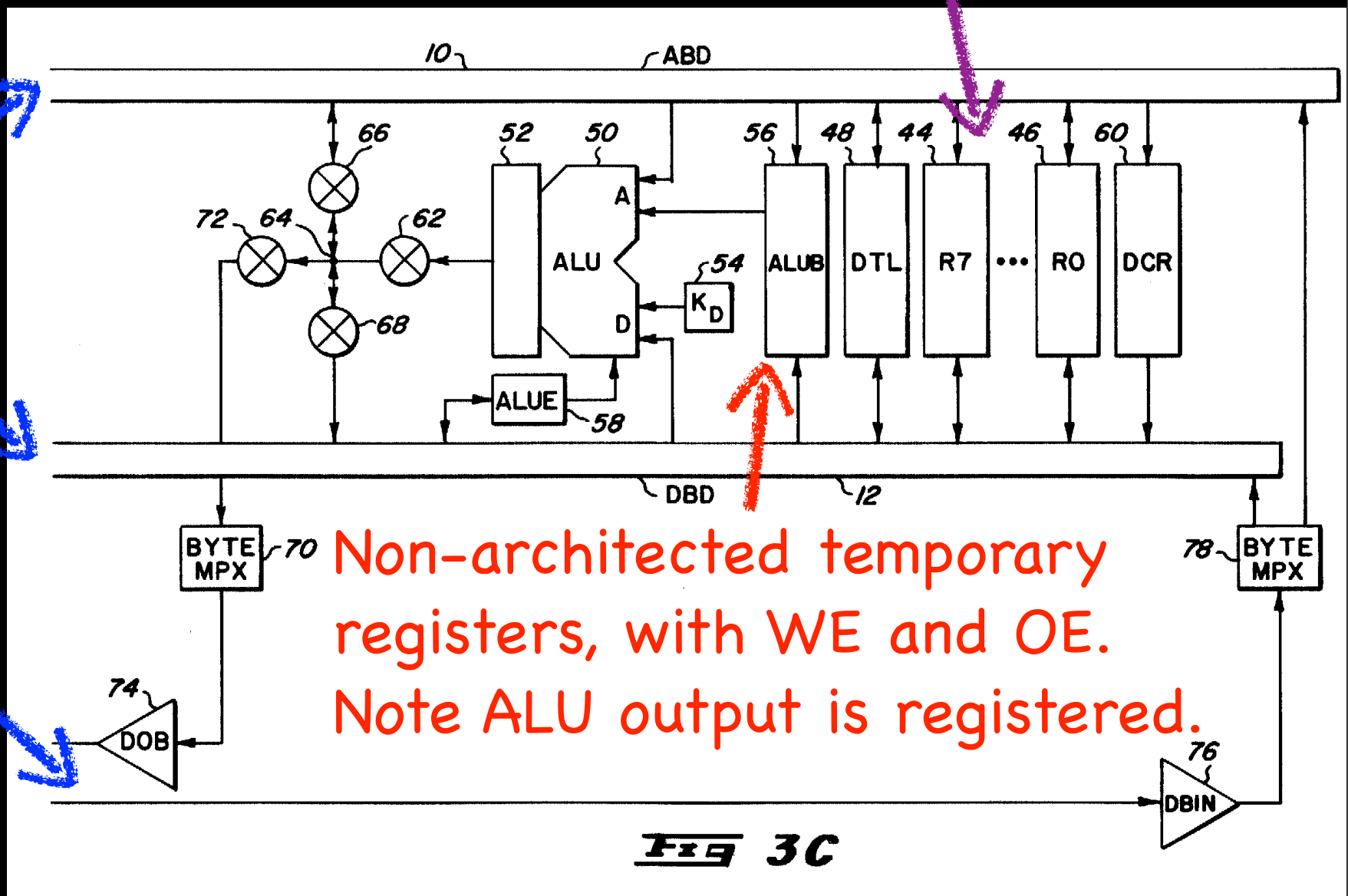


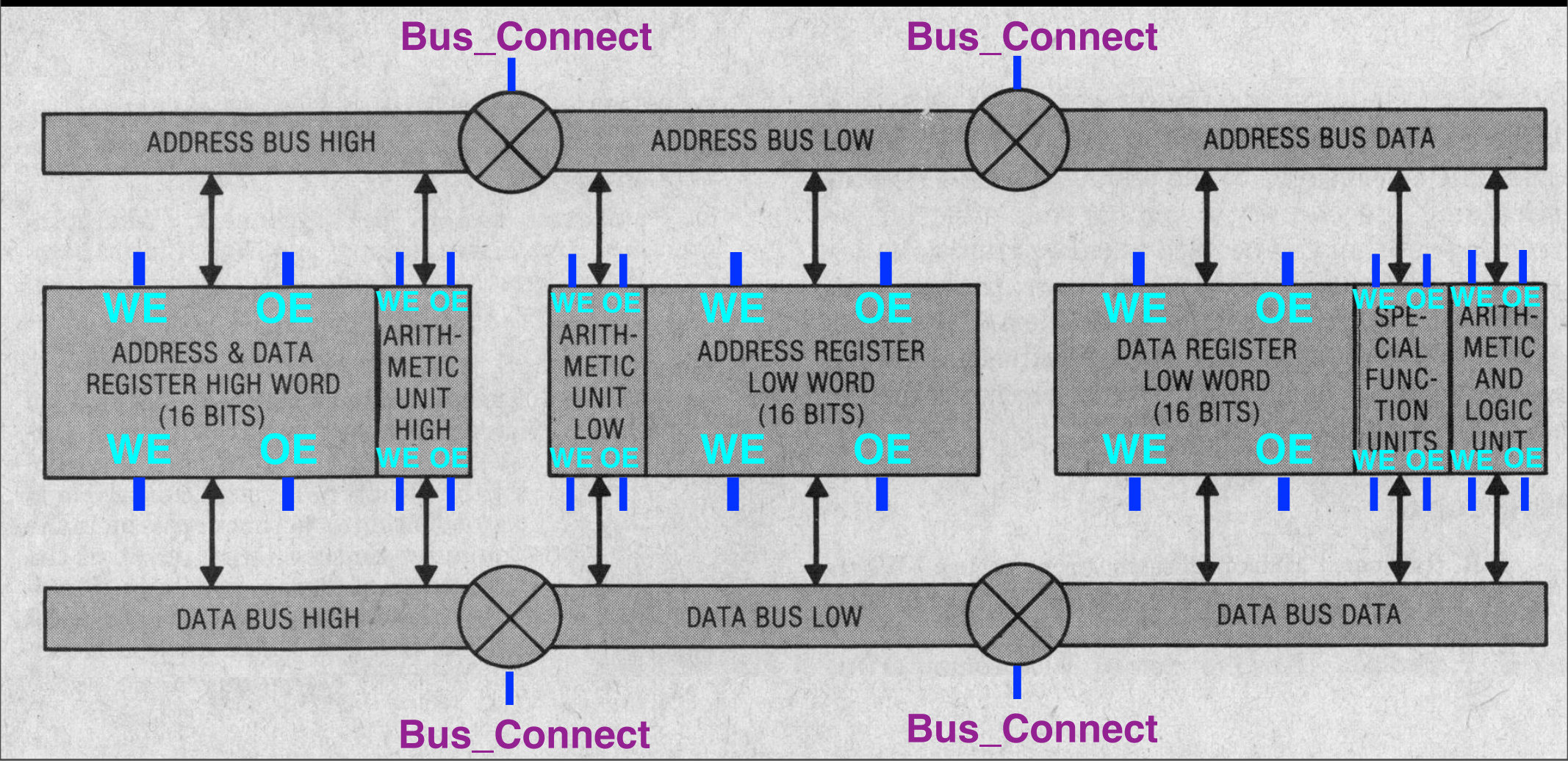
Fig 3C

Non-architected temporary registers, with WE and OE.
Note ALU output is registered.

68000: Complete datapath

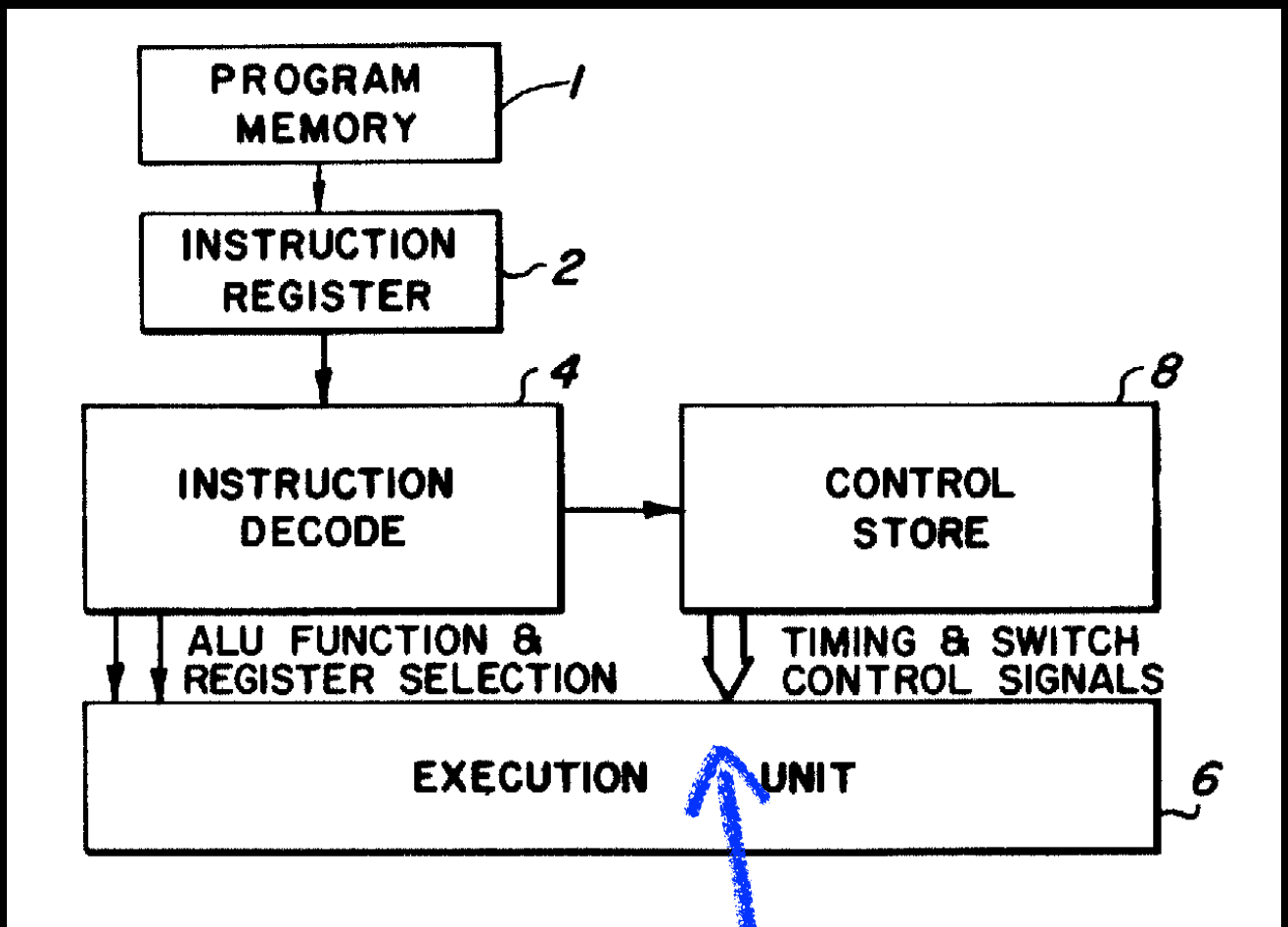
Controller executes an instruction by toggling enables over several cycles.

Six 16-bit on-chip buses, with registered ALUs.
Enable signals (WE, OE) lets ALUs and registers write or listen to top or bottom bus in its domain.
Switches optionally connect certain buses.



Controller: 10,000 foot view

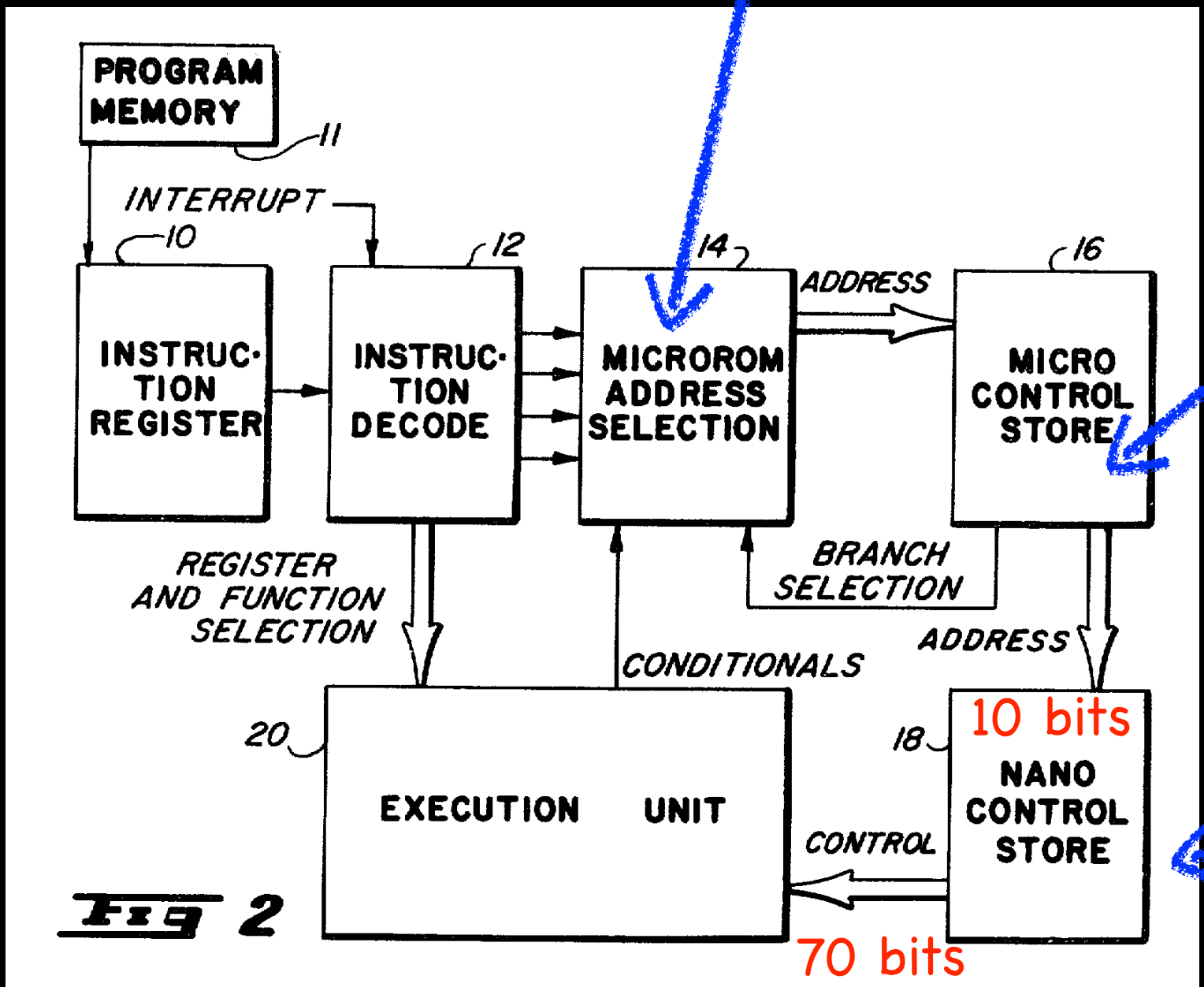
Control signals that are constant for all cycles of instruction.



Control signals that the microcode engine toggles to do the work.

Controller: 1000 foot view

Microcode program counter and branch control logic.



Multi-cycle microcode programs for each instruction in the ISA

Decodes 10-bit states into 70-bit datapath signals

Controller: 100 foot view

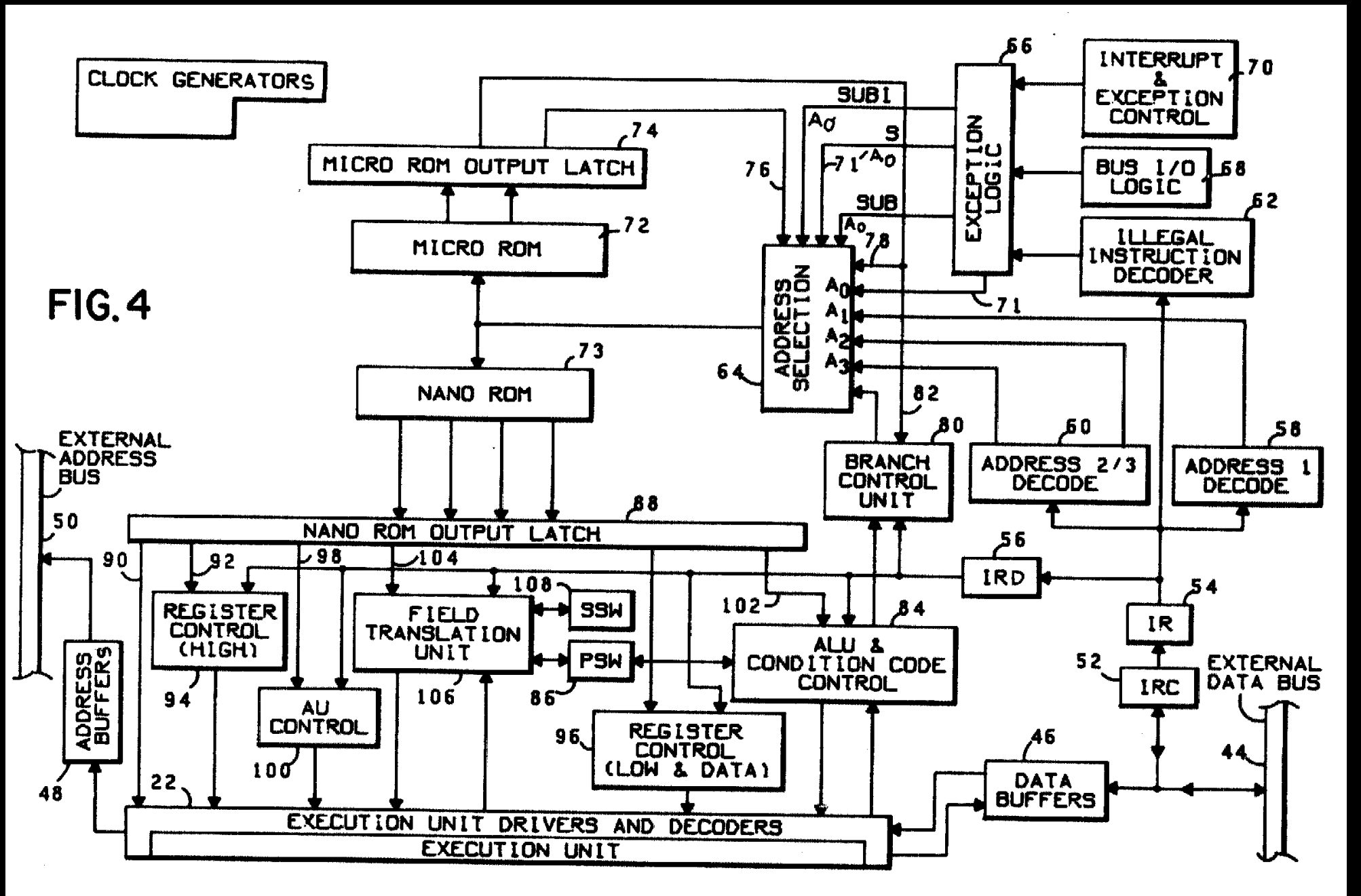


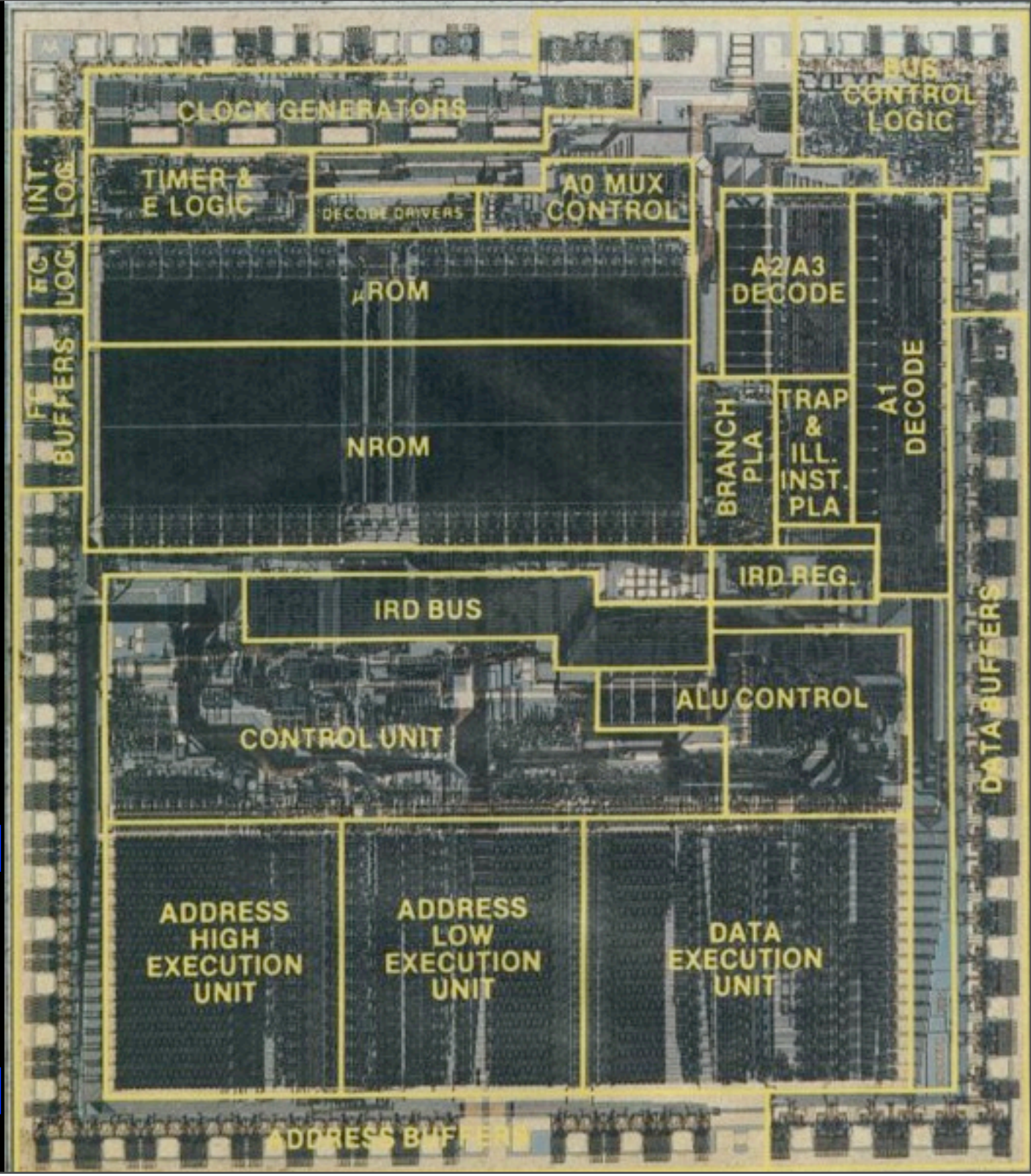
FIG. 4

68000 die photo

Microcode Controller:

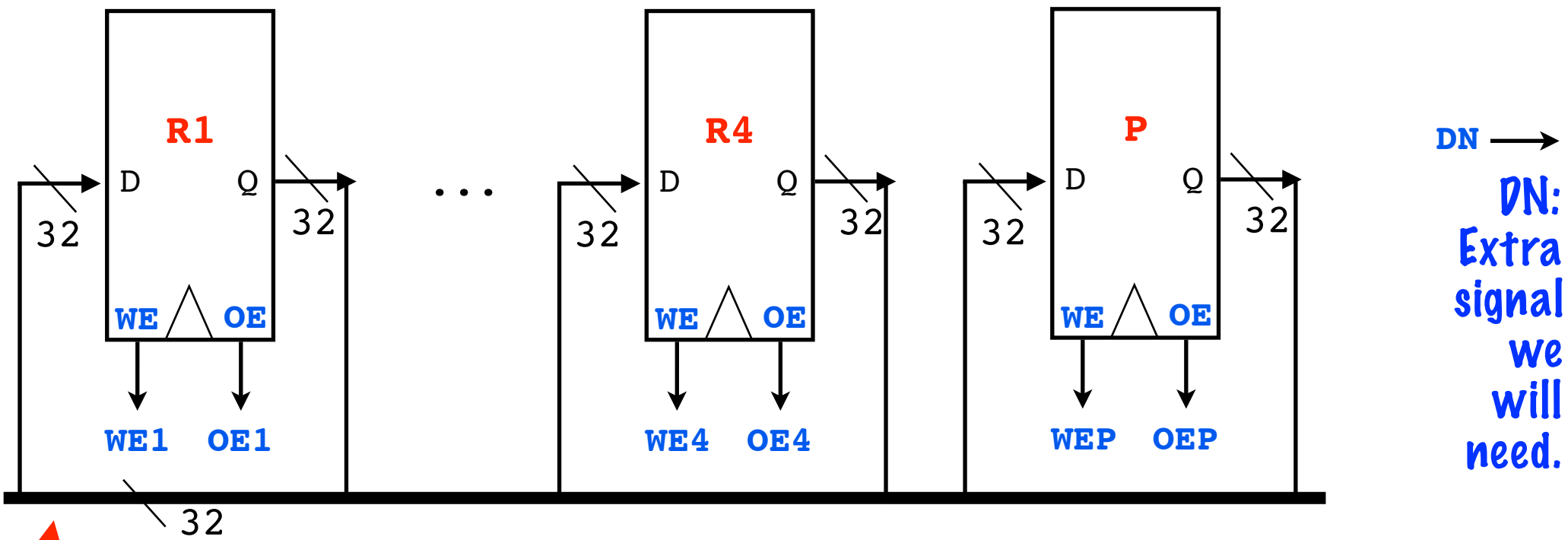
Two large Read-Only Memories (ROMs) and many small state machines.

Datapath: 6 buses, 70 control signals



But ... what exactly is microcode?

Here is a simple CPU data path.
4 architectural registers (R1 - R4) and a temporary (P).



All registers use output enable (OE) and write enable (WE) signals to transfer state values.

A bus: 32 wires that one register per cycle can write (all can read bus)

But ... what exactly is microcode?

Each clock cycle, we can think of this data path as executing an 11-bit microcode instruction word:

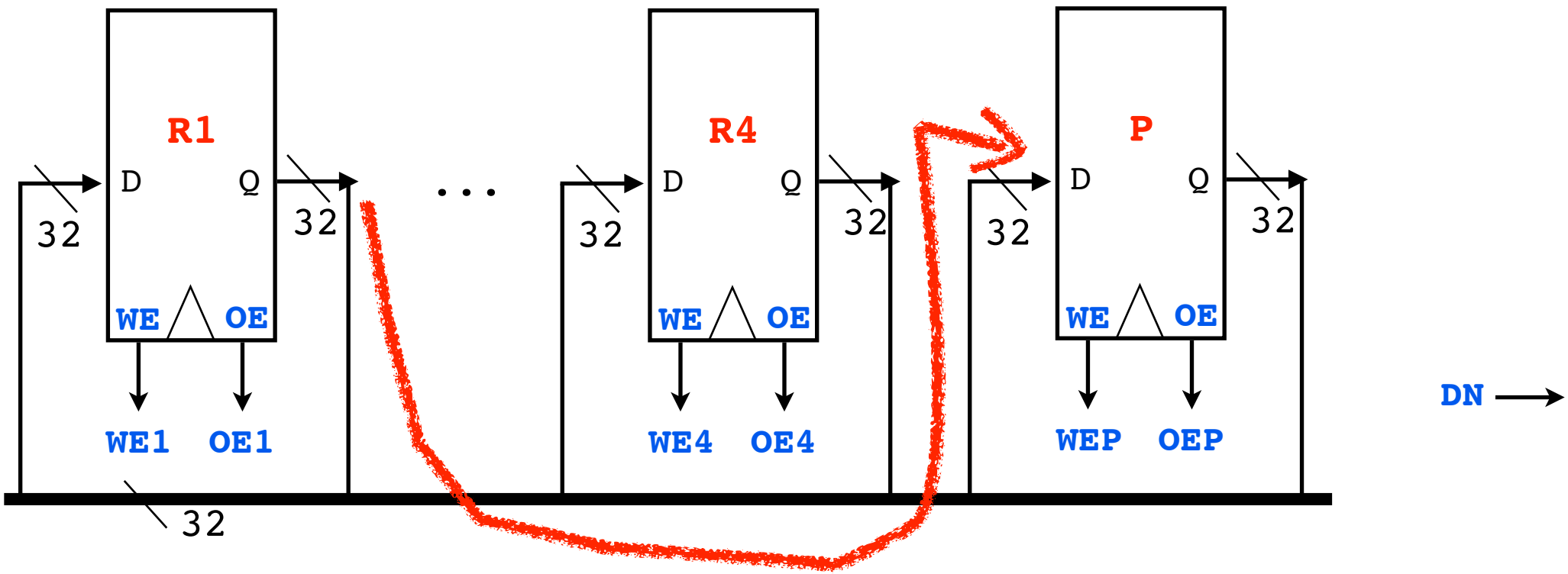
One microcode instruction - binary format

DN	WE1	WE2	WE3	WE4	WEP	OE1	OE2	OE3	OE4	OEP
0	0	0	0	0	1	1	0	0	0	0

Assembler format

```
OE1, WEP;
```

a list of "1" columns.



Macro instructions and microcode

A "macro" instruction: SHUFFLE

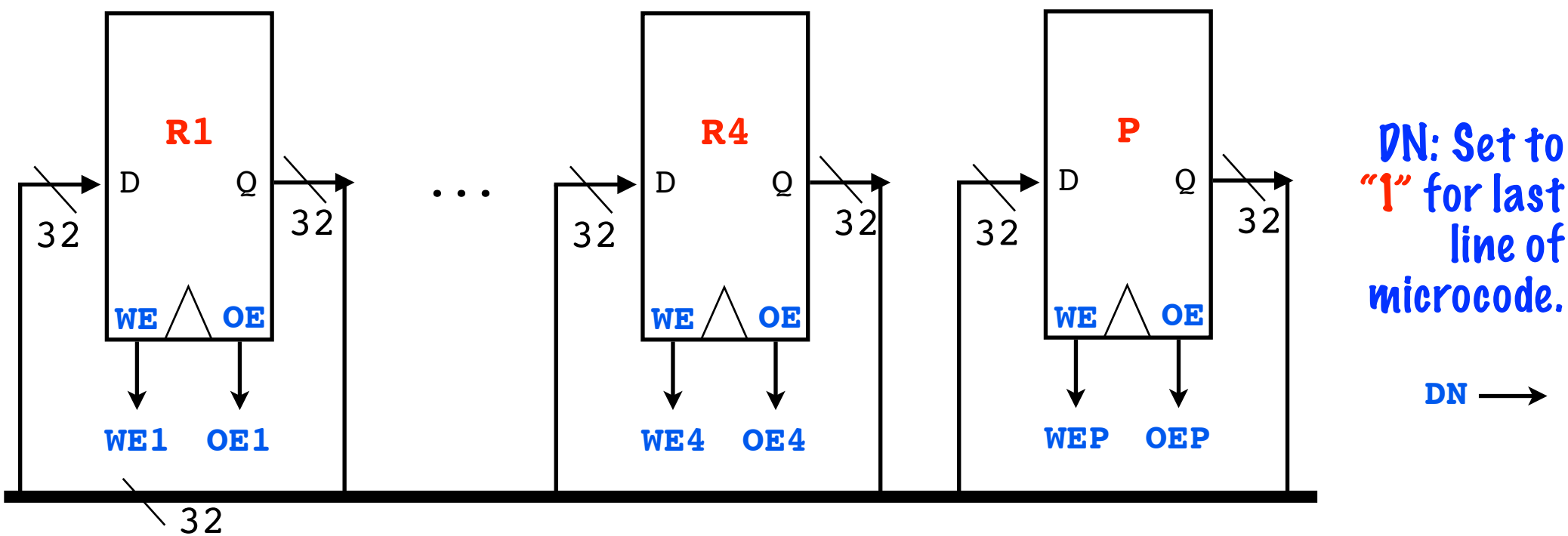
SHUFFLE semantics:

After execution of SHUFFLE:
R1 takes on the old value of R2,
R2 takes on the old value of R3 ...
R4 takes on the old value of R1.

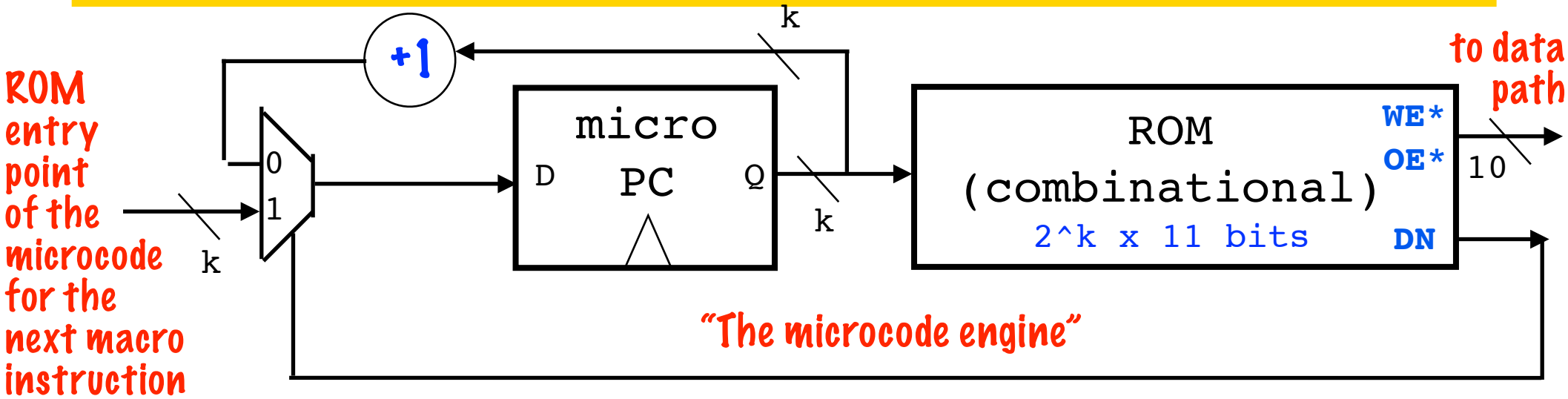
Microcode implementation:

```
OE1, WEP;  Save R1 in P.
OE2, WE1;  R1 <- R2
OE3, WE2;  R2 <- R3
OE4, WE3;  R3 <- R4
OEP, WE4, DN;  R4 <- P
```

Takes 5 cycles to execute.



Assembling the microcode for SHUFFLE



Binary: Stored in read-only-memory (ROM)

Assembler:

```
OE1,WEP;   Save R1 in P.
OE2,WE1;   R1 <- R2
OE3,WE2;   R2 <- R3
OE4,WE3;   R3 <- R4
OEP,WE4,DN;R4 <- P
```

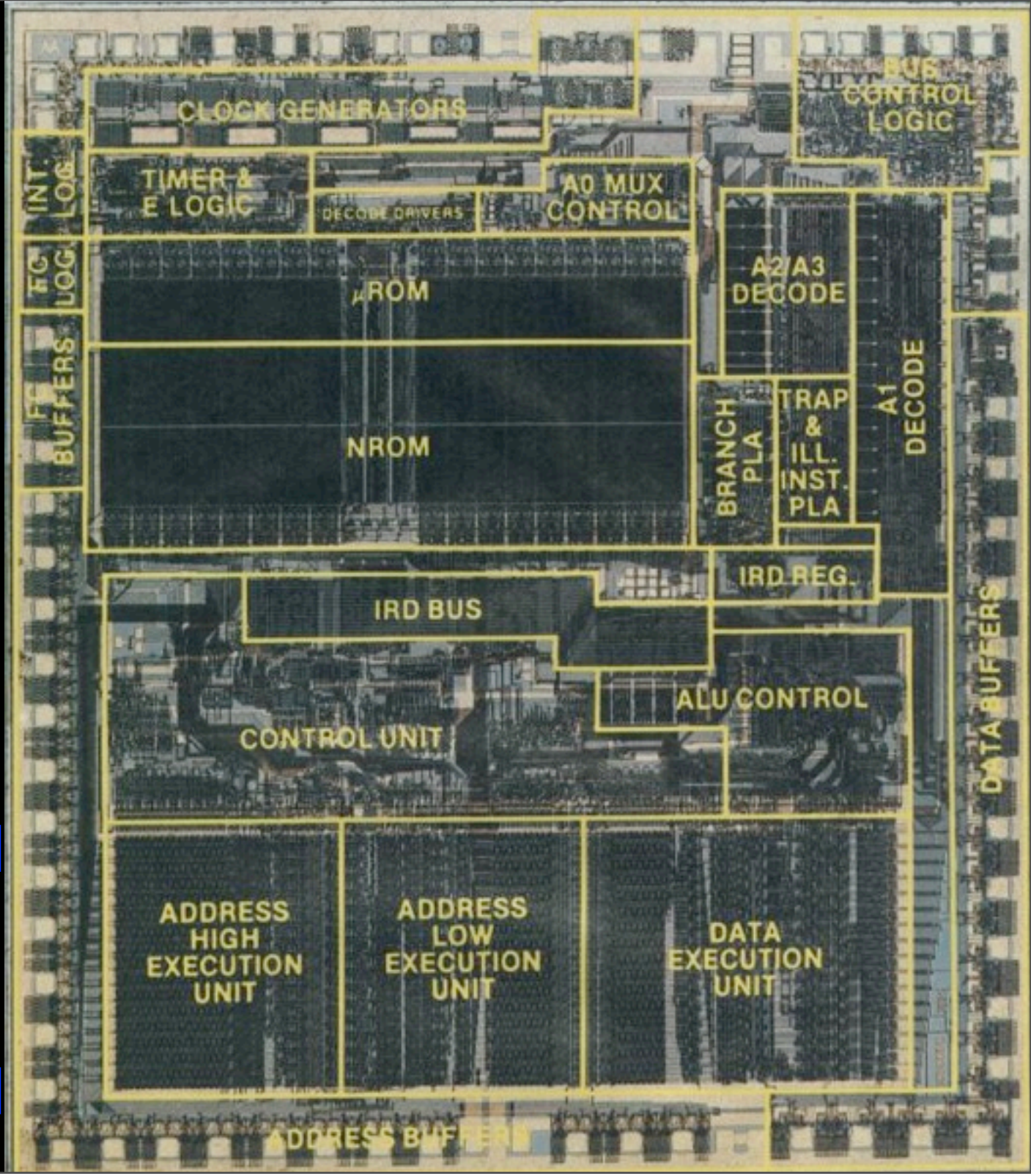
DN	WE1	WE2	WE3	WE4	WEP	OE1	OE2	OE3	OE4	OEP	
0	0	0	0	0	1	1	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	0	1	0	0	2
0	0	0	1	0	0	0	0	0	1	0	3
1	0	0	0	1	0	0	0	0	0	1	4

microcode instruction addresses in ROM

How do we design a CPU that executes 58-cycle instructions?

Microcode controller: Takes up most of chip area.

Datapath: Specialized for multi-cycle operation.



Intel Quark ...

note ☆ stop following 28 views

CSUA + Intel Code for Good Hackathon

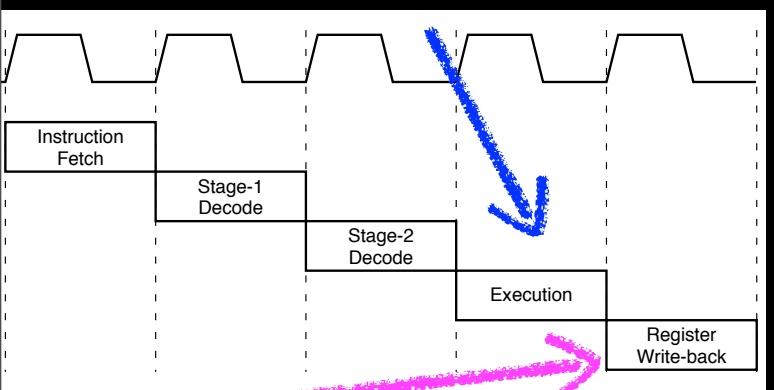
When: Saturday, 2/15, 5pm to Sunday, 2/16, 5pm
Where: Wozniak lounge in Soda

There is an exciting 24-hour hardware + software hackathon revolving around the innovative new Intel Galileo Development board! Intel will be providing Berkeley students with 10 Galileo boards, complete with dev kits (cables, Arduino Uno R3 sensors, and more). Each team of 4-5 students will be able to work with their own board. Here's a link with more info about the board http://www.intel.com/content/www/us/en/do-it-yourself/galileo-maker-quark-board.html

Register at http://bit.ly/1behwHc. Space is limited and growing fewer as you read this!

Intel Quark: "New" microcoded Pentium CPU.

Many instructions have only one line of microcode.



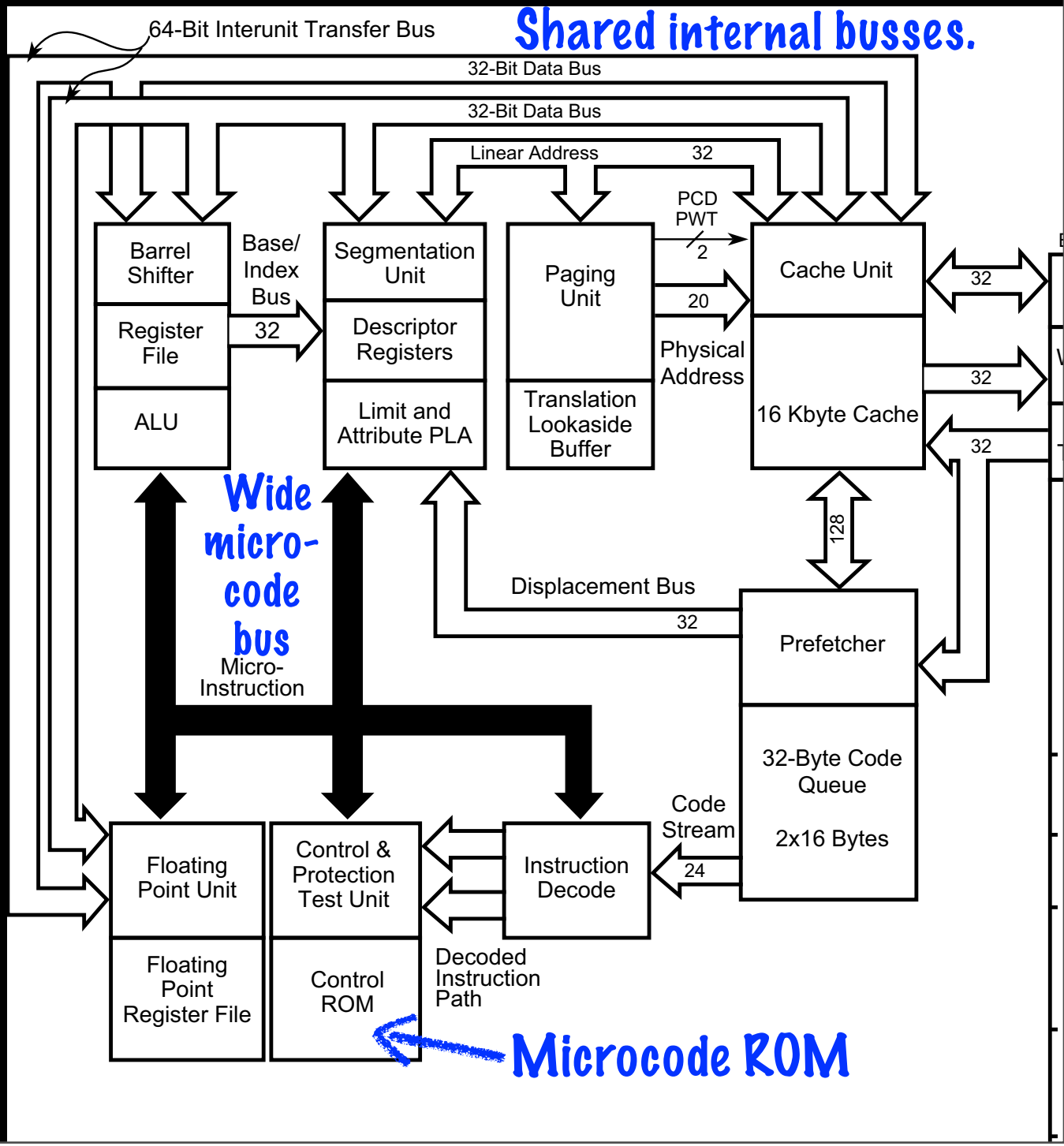
Hazards? Stall ...

Multi-line instrs stall in EX stage.

```
F3 | F4 | F5 |          F6 | F7
D2 | D3 | D4 |          D5 | D6
E1 | E2 | E3 | E4   E4   E4 | E5
```

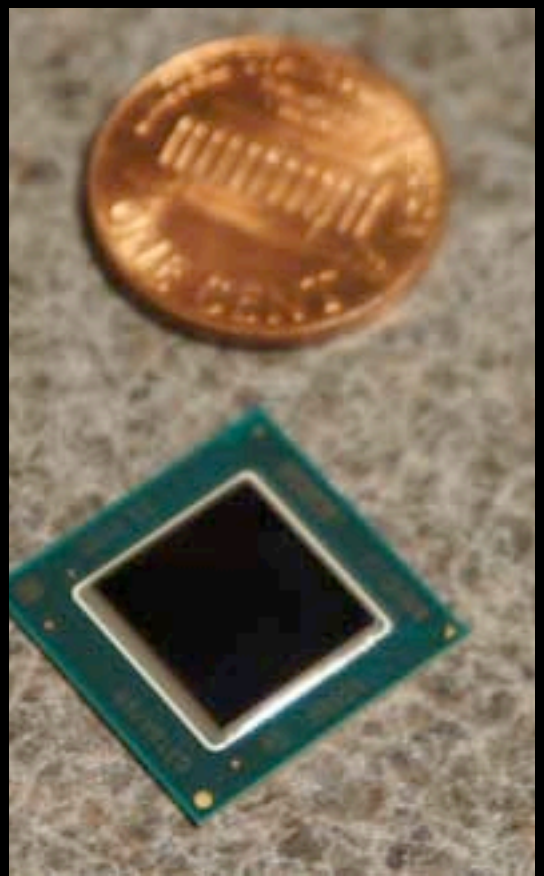
Figure 4: Simplified instruction execution sequence

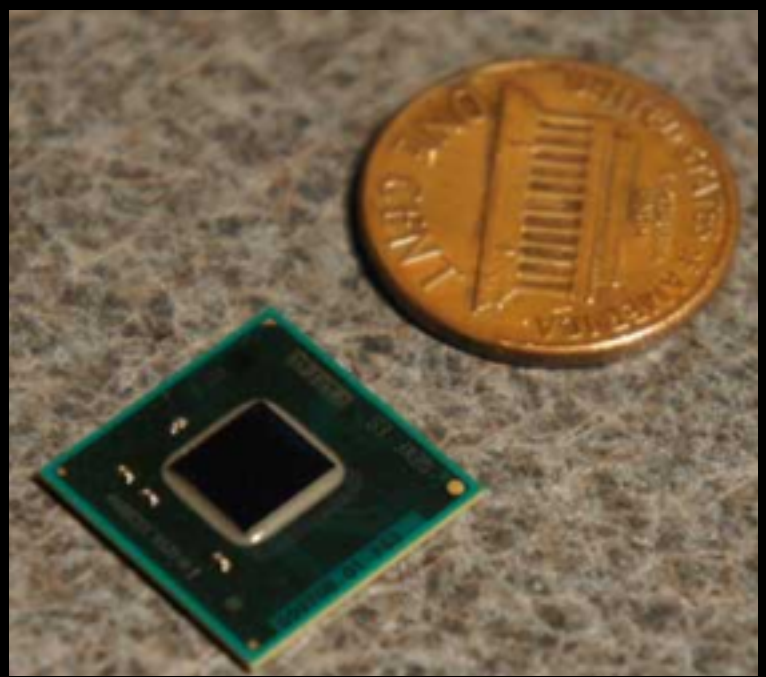
Microcode in 2014?



Aimed at "Internet of Things" - very low cost.

Bay Trail Z3000. New Intel "Atom" chip for low-cost tablets. 22 nm process.





Quark: a $5 x86. 20% of Z3000 die-size 32 nm process.

Based on a 1994 486DX4 design.

Challenge: Quark runs a standard IA-32 instruction set, but has 1/10th the transistors.

Solution: Tiny caches (16 KB vs Z3000 multi-MB), no GPU, ... and, microcode CPU (vs. 17-stage, dual-issue, out-of-order, multi-core Z3000 CPU).

Meeting IoT price points with a more modern design

Best Sellers in Electronics

1.

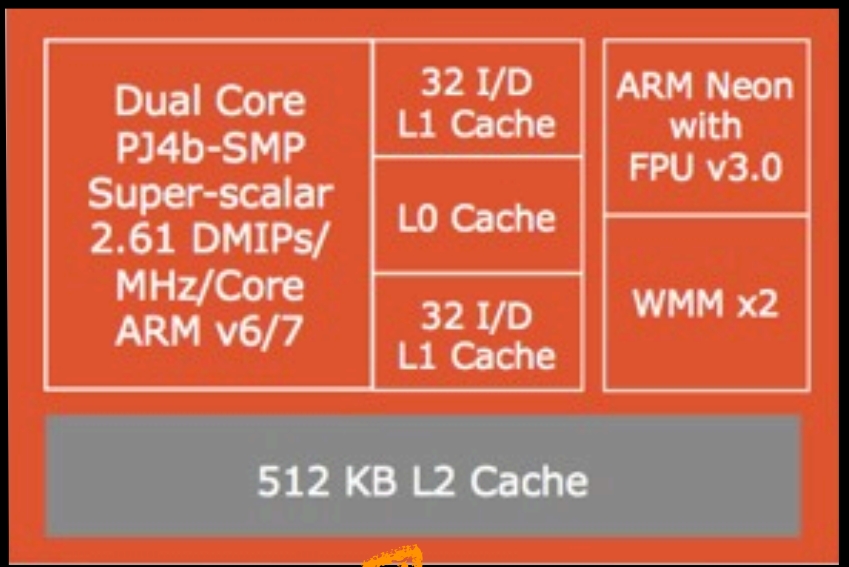
$35 retail implies Bill of Materials (BOM) in the $20 range ...

Google Chromecast HDMI Streaming Medi...

★★★★☆ (11,584)

$35.00

200 used & new from $29.99

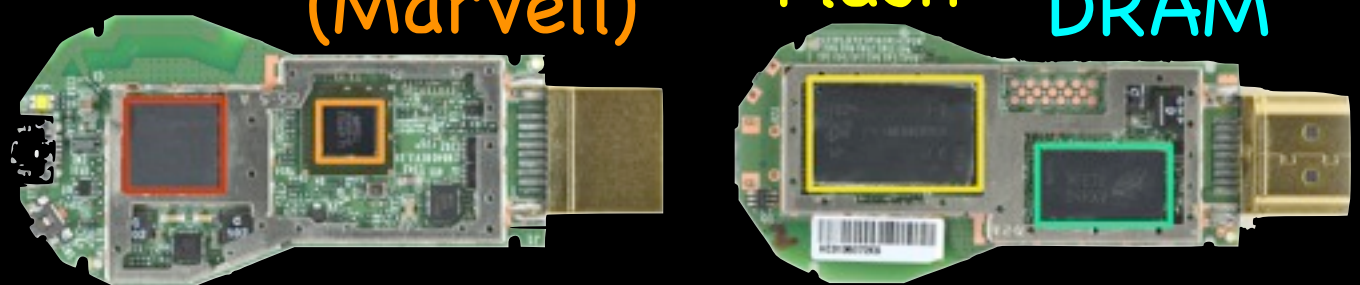


Wi-Fi

ARM CPU (Marvell)

2 GB Flash

512 MB DRAM



Chromecast:

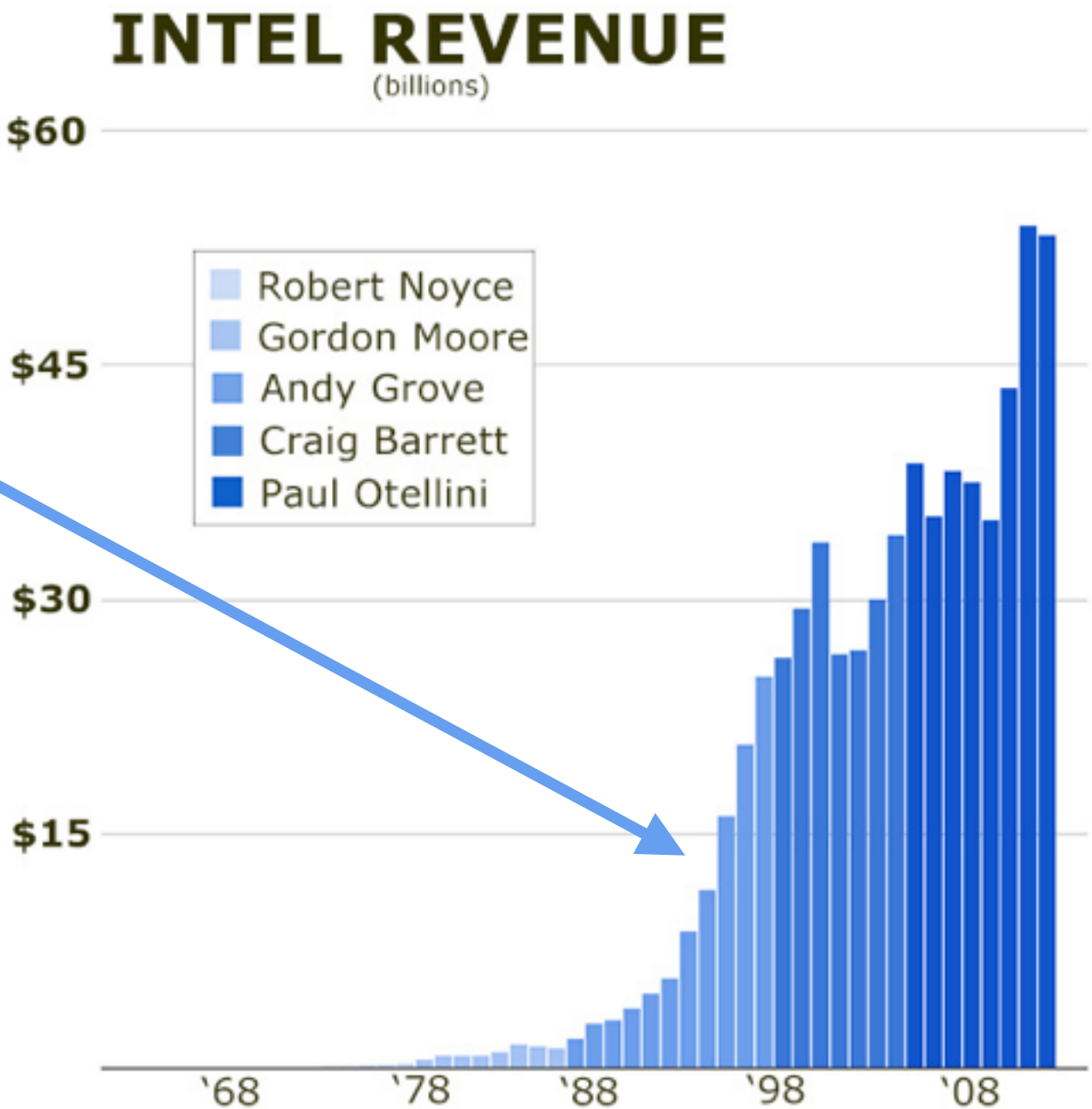
Web browser in a flash-drive form factor. Plugs into the HDMI port on a TV. Includes a Wi-Fi chip so you can control the browser from your cell phone.

Break

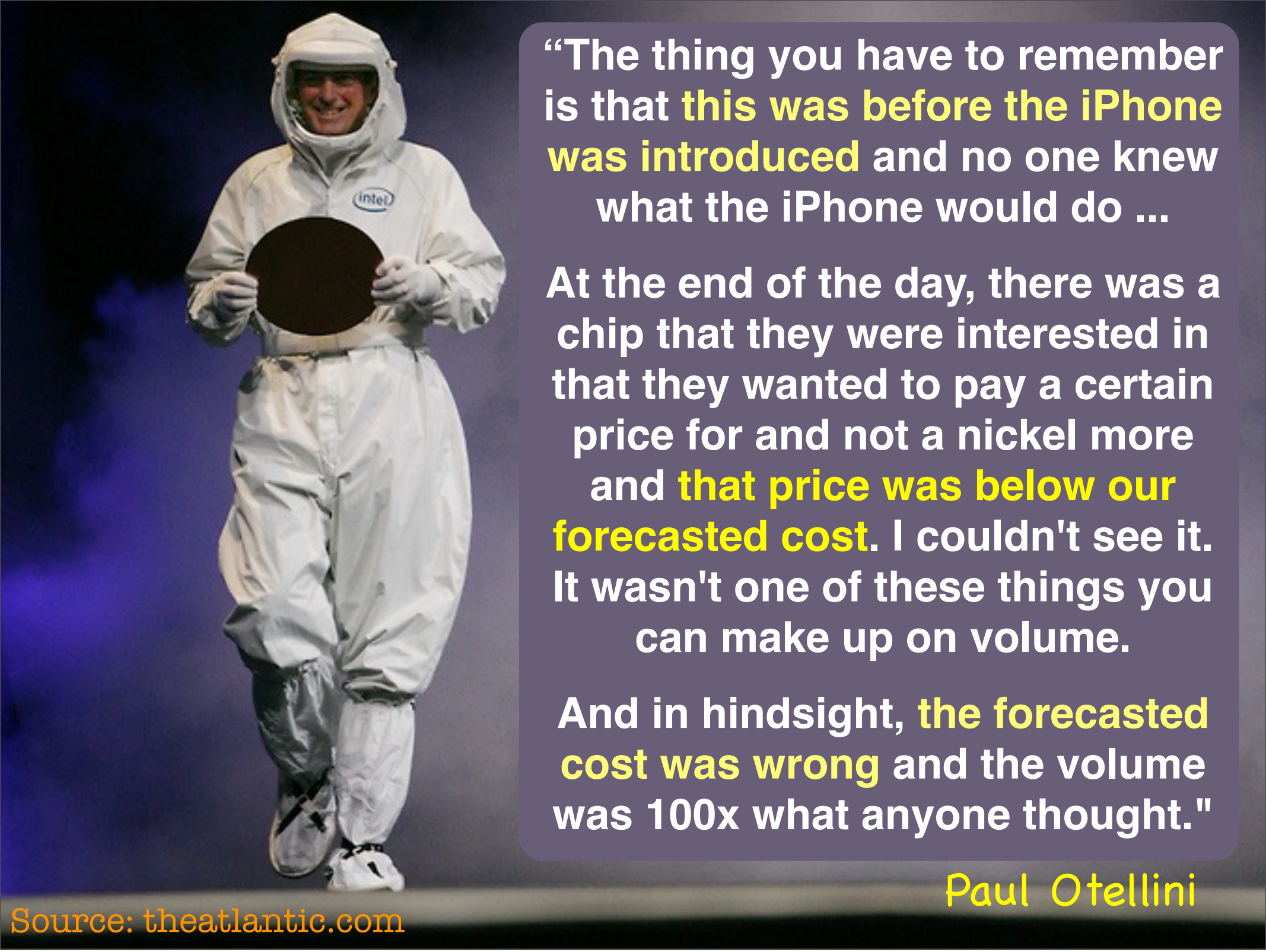
Play:

Economics ... inextricably part of chip design

Andy Grove
UCB Ph.D. 1963




Moscone Center, 2005.
Apple switches
the Mac to Intel CPUs.
intel
Steve Jobs
Paul Otellini,
Haas School
MBA 1974



"The thing you have to remember is that this was before the iPhone was introduced and no one knew what the iPhone would do ...

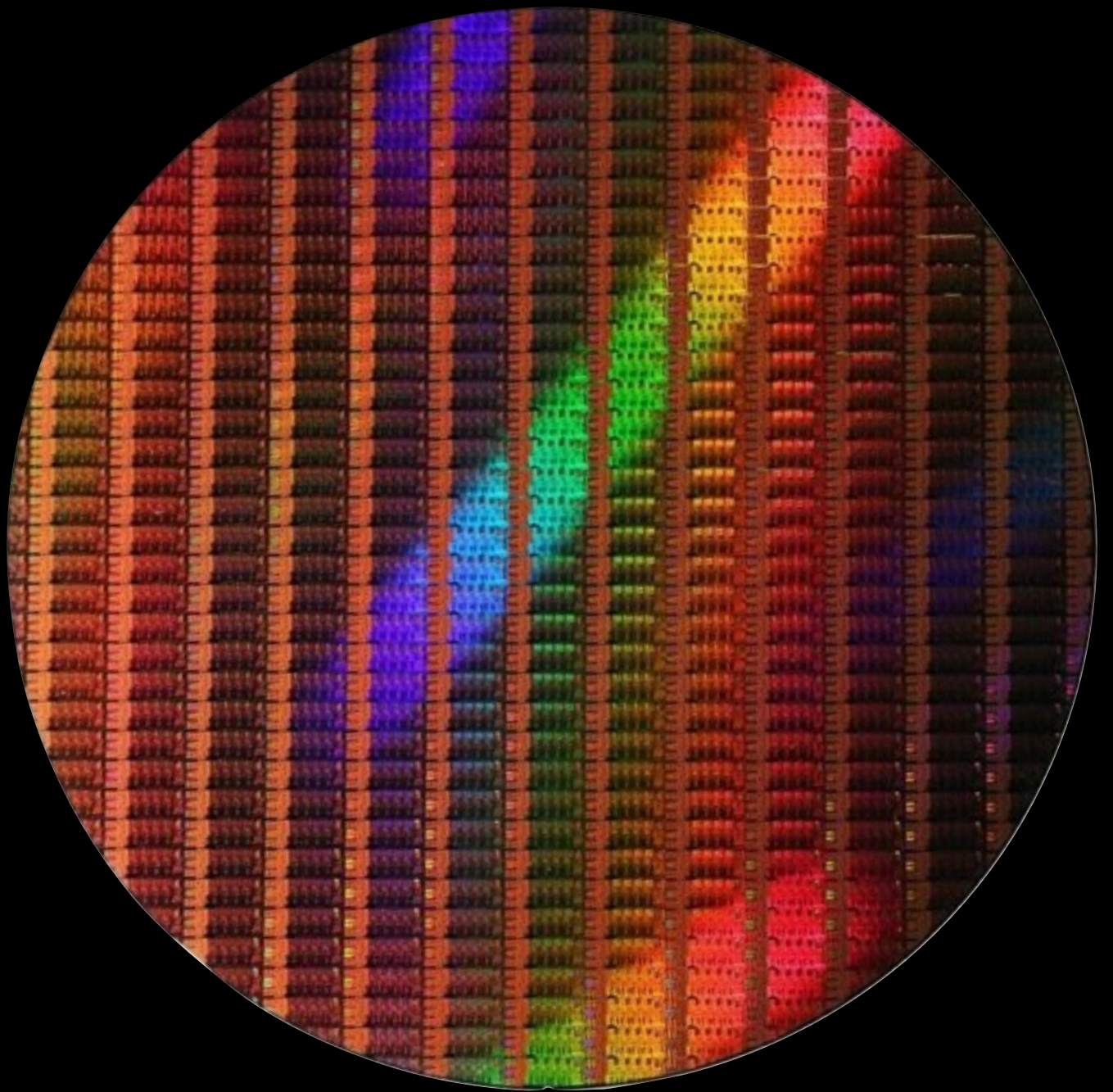
At the end of the day, there was a chip that they were interested in that they wanted to pay a certain price for and not a nickel more and that price was below our forecasted cost. I couldn't see it. It wasn't one of these things you can make up on volume.

And in hindsight, the forecasted cost was wrong and the volume was 100x what anyone thought."

Paul Otellini

Source: theatlantic.com

Intel Haswell wafer (22nm FinFET)

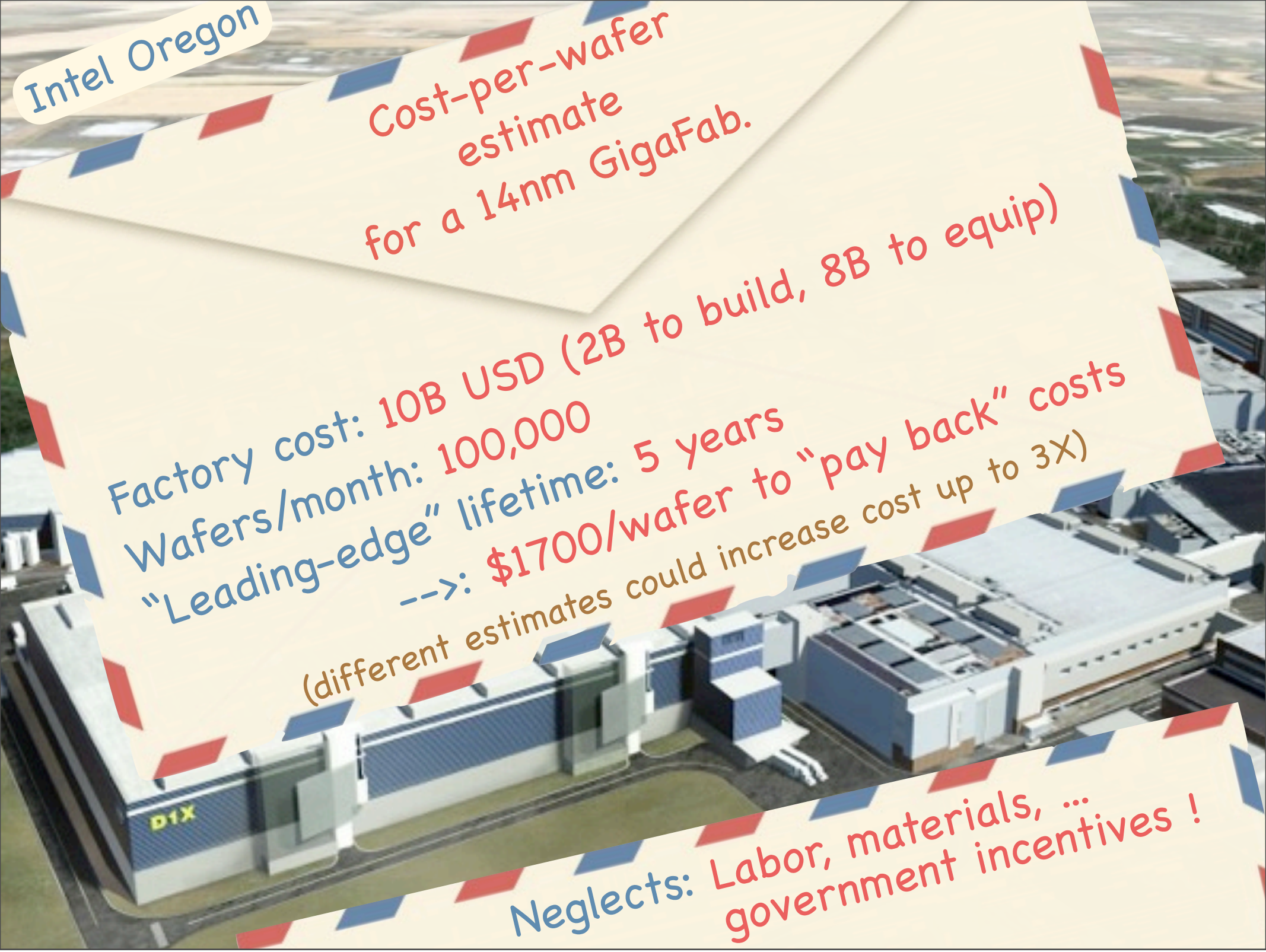


How do you estimate the manufacturing cost of a chip?

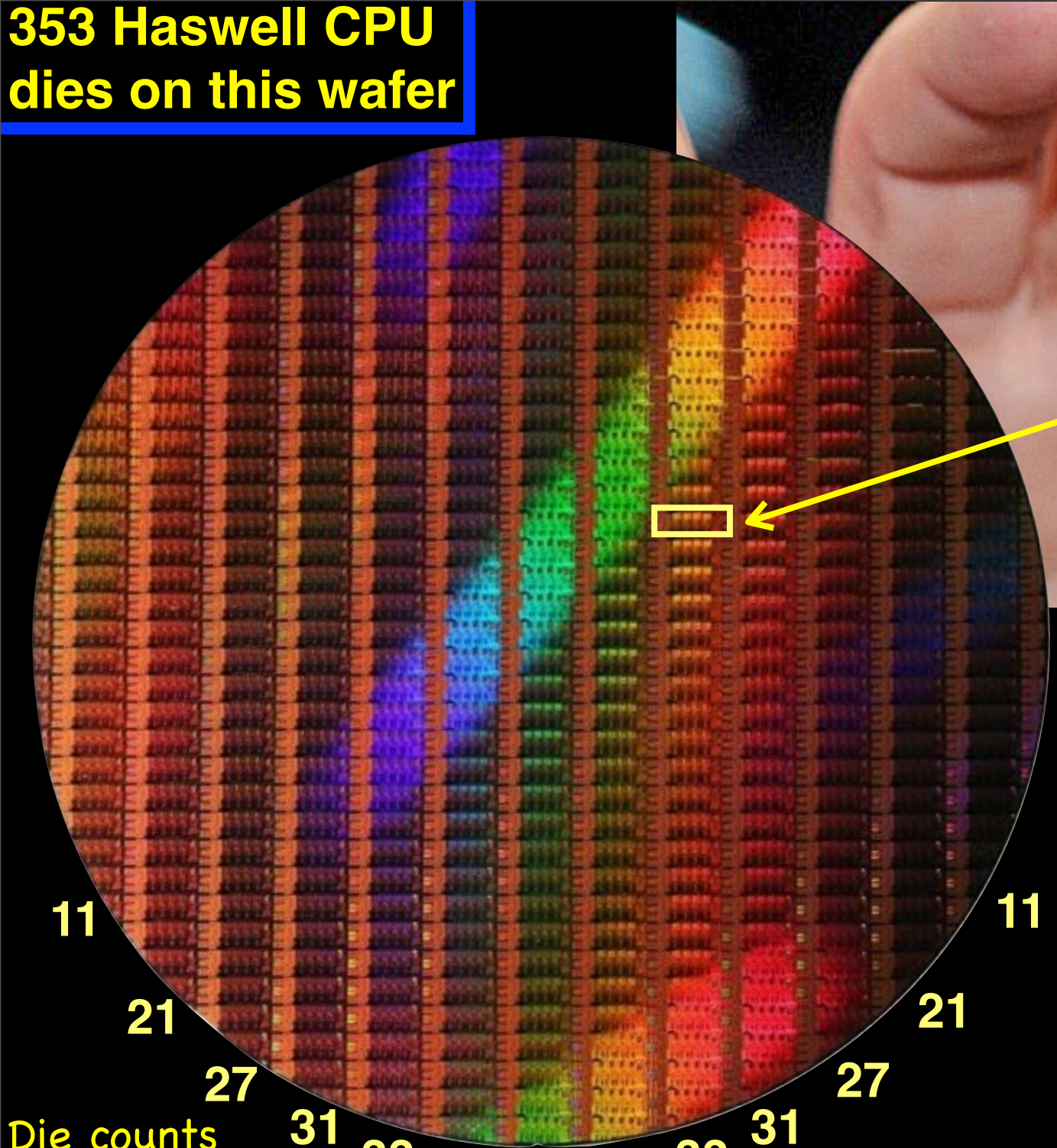
Chip factories ("fabs") process wafers that contain many copies of a chip ("dies")

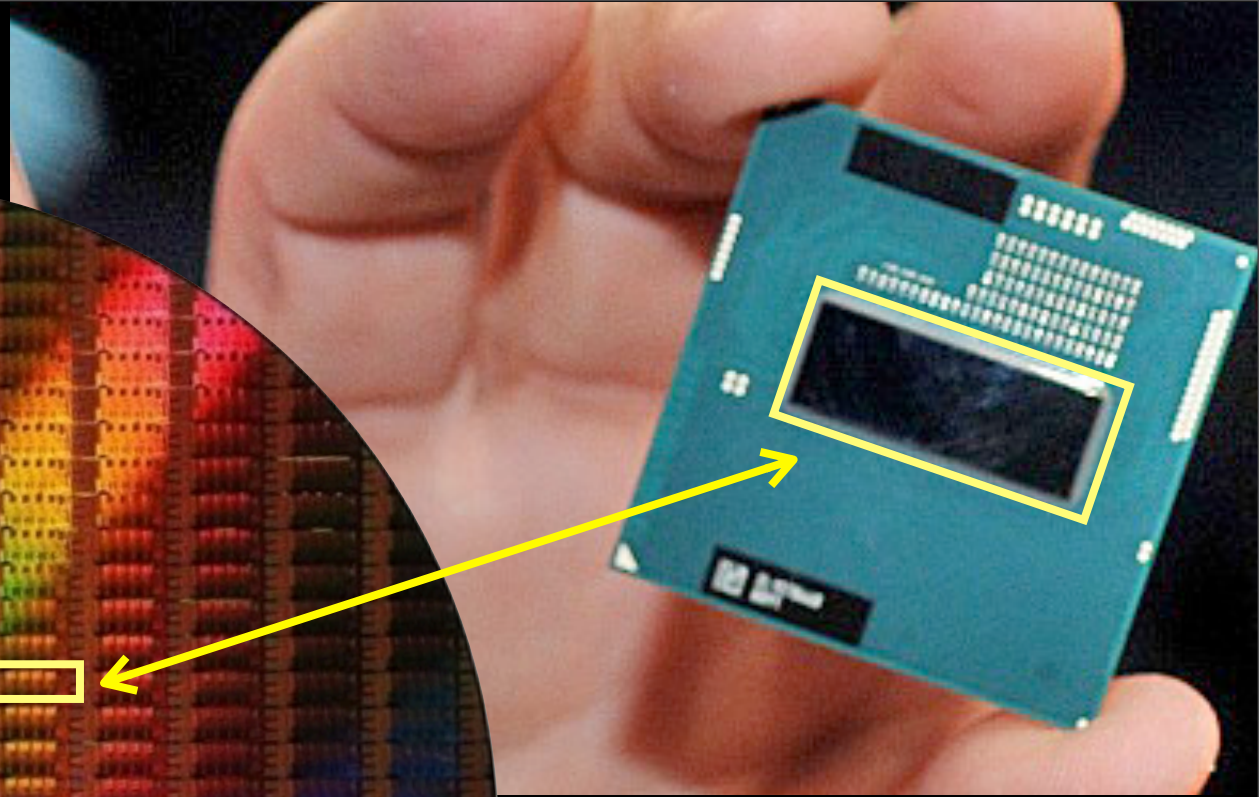
So, step one is estimating the cost to manufacture a wafer.


Intel Oregon
D1X


Intel Oregon
Cost-per-wafer estimate for a 14nm GigaFab.
Factory cost: 10B USD (2B to build, 8B to equip)
Wafers/month: 100,000
"Leading-edge" lifetime: 5 years
-->: $1700/wafer to "pay back" costs
(different estimates could increase cost up to 3X)
Neglects: Labor, materials, ... government incentives !
D1X

353 Haswell CPU dies on this wafer





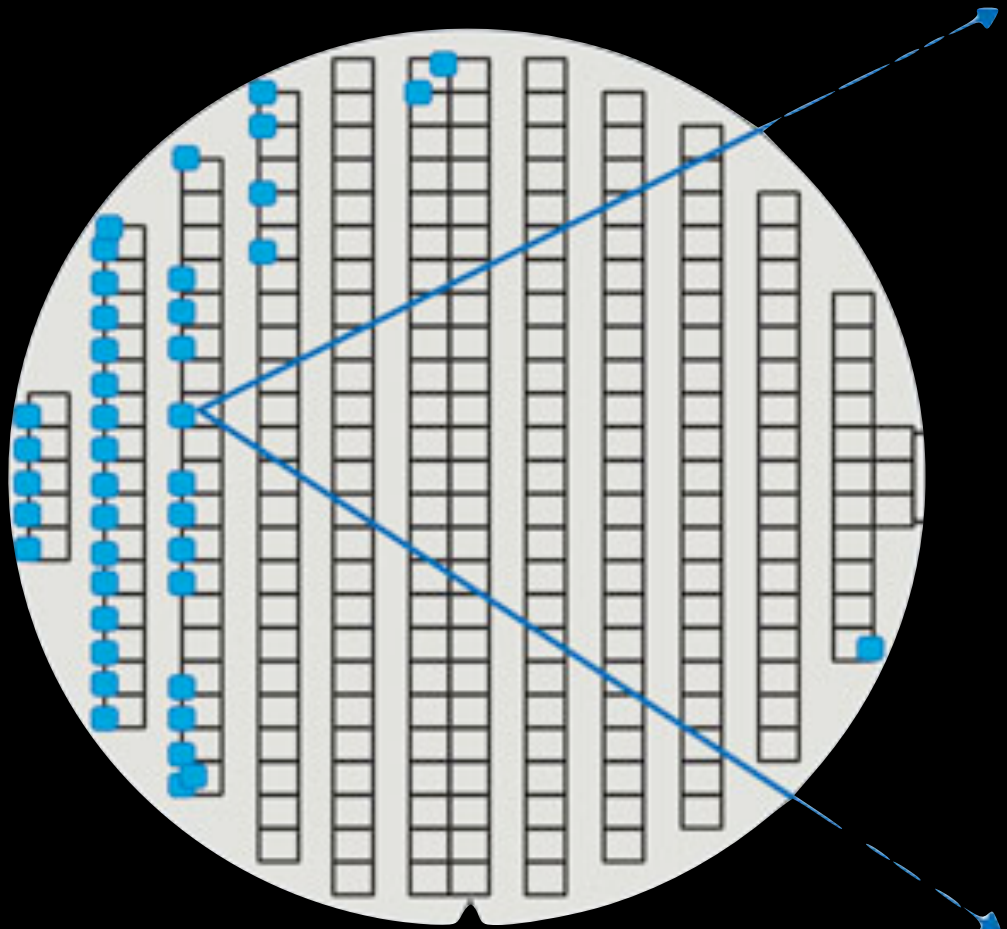
==> $4.80 per die!

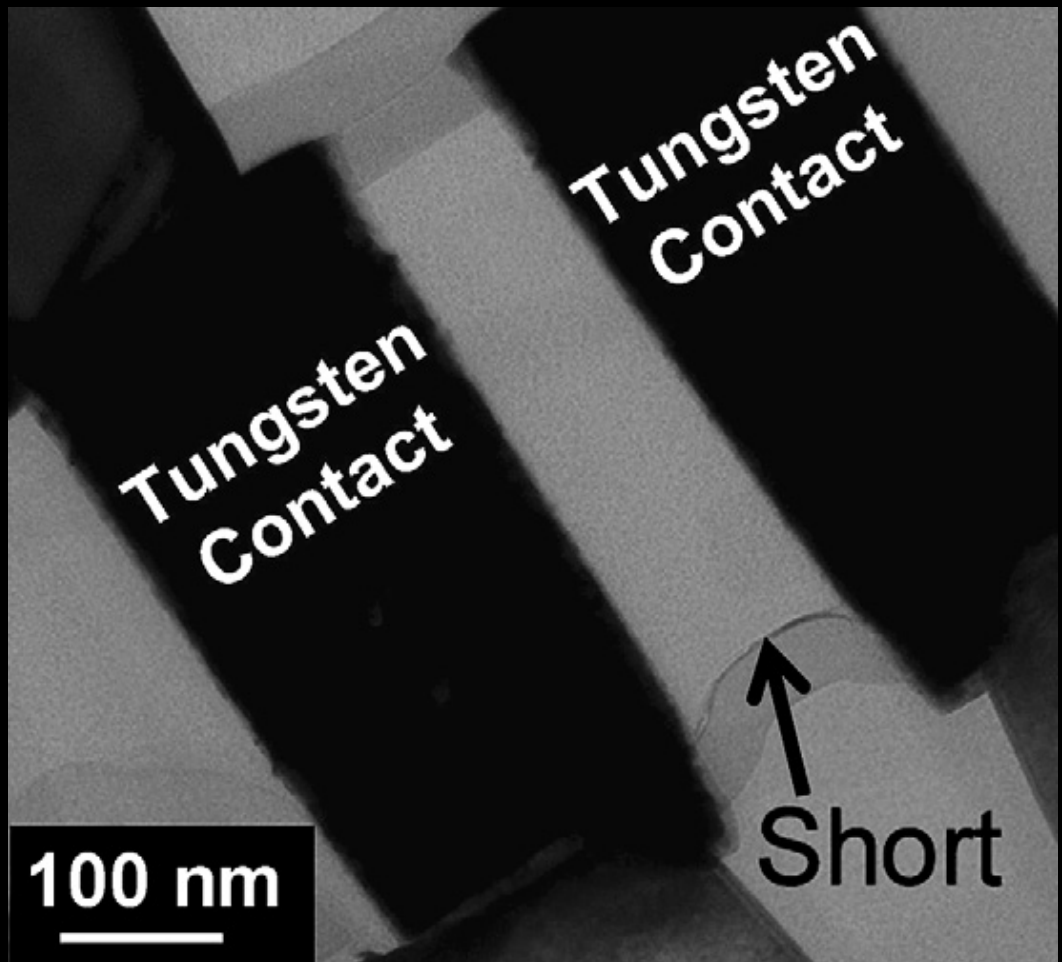
But if die were twice as big ...
$9.60 per die!

This is one reason why die size matters.

This analysis is optimistic ...

Yield: Not all dies on the wafer will work!





A = die area D_o = defects per unit area

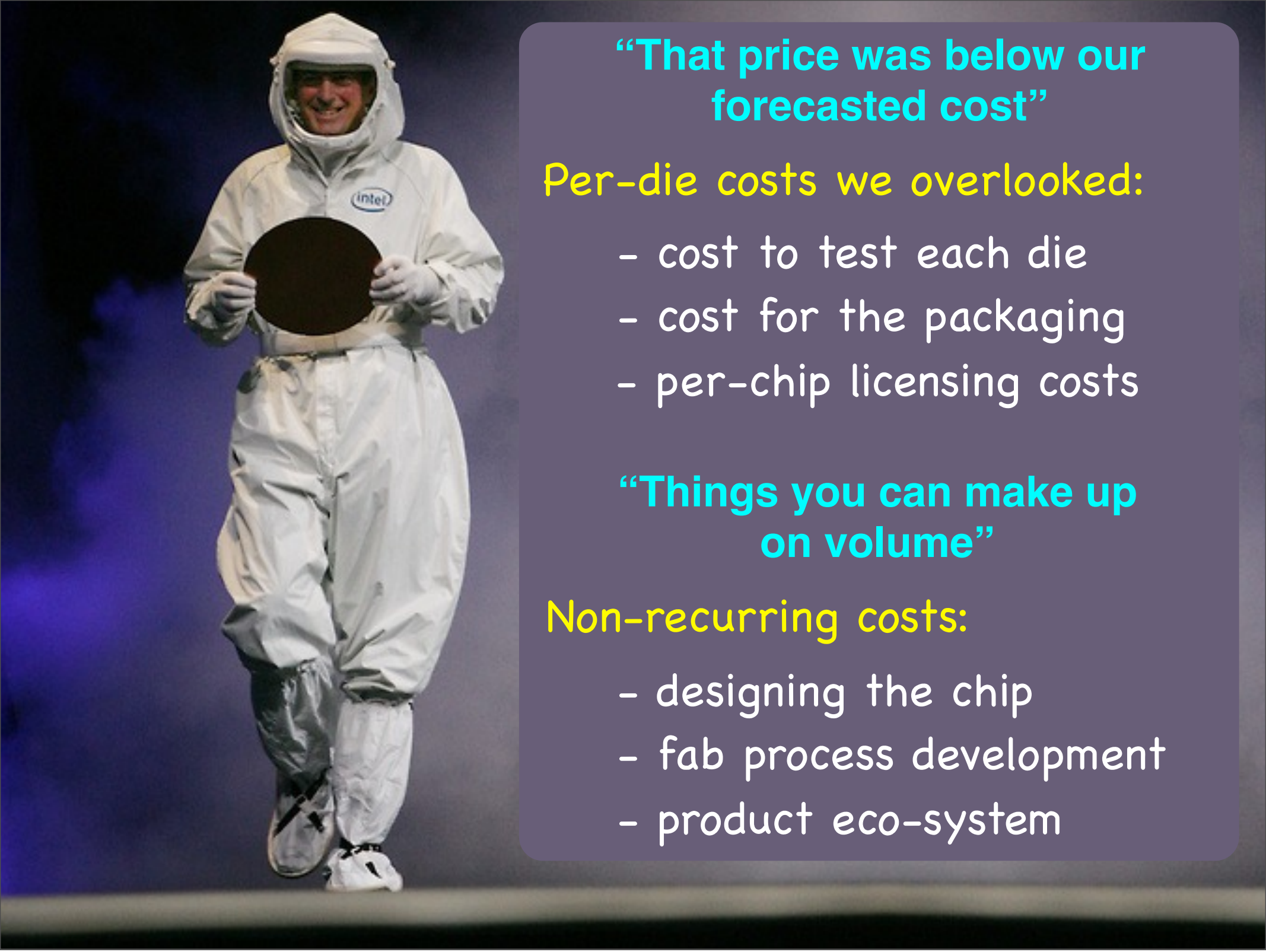
Yield (A) = 100% × exp(-A*D_o)

If A << D_o, yield approach 100%.

If A ≈ 2 D_o, yield is about 13%

Another reason why die size matters!

Note: this "Poisson" model is overly pessimistic, real-world yield models are more complex.



"That price was below our forecasted cost"

Per-die costs we overlooked:

- cost to test each die
- cost for the packaging
- per-chip licensing costs

"Things you can make up on volume"

Non-recurring costs:

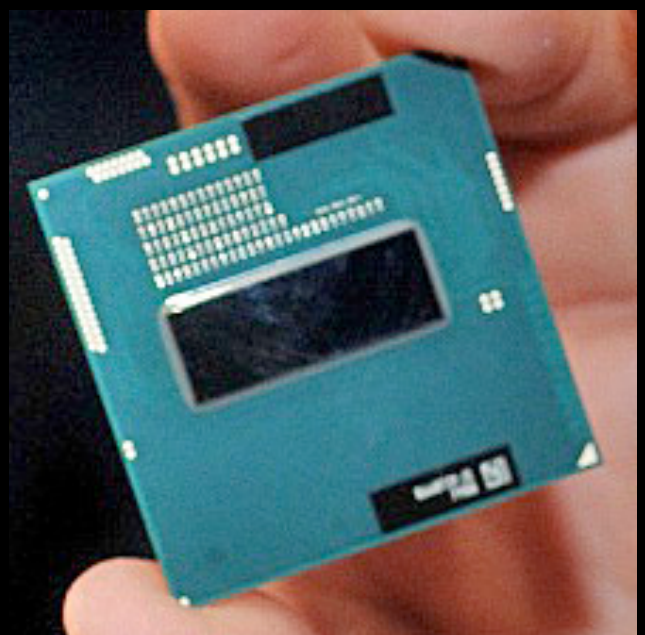
- designing the chip
- fab process development
- product eco-system

Industry Organization

i286 design team (1984)

Integrated Device Manufacturers (IDMs)

Intel designs "standard product" chips that sell to many customers (Apple, Dell, HP).

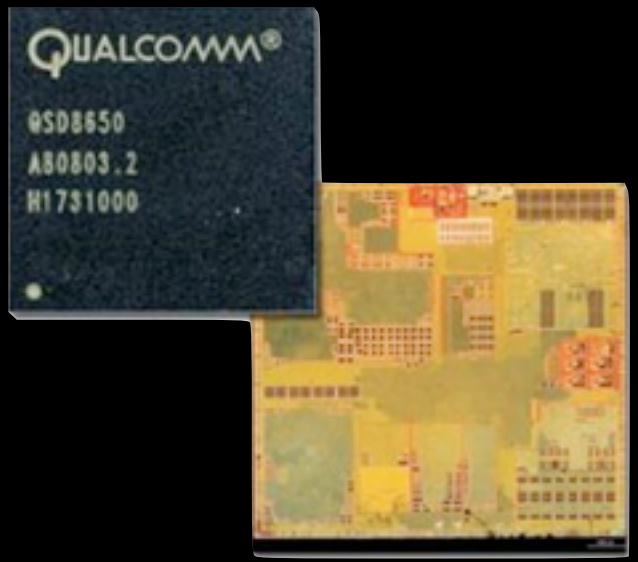


This was the original industry model.

It develops IC process technology, for use in its own wafer fabs.

It avoids "competing with its customers" (Intel doesn't make notebook PCs).

Fabless Merchant Model



Qualcomm designs chips for use in smartphones, but does not own fabs.

TSMC (a "foundry") owns fabs, but does not do chip design. Qualcomm contracts with TSMC to design its chips.

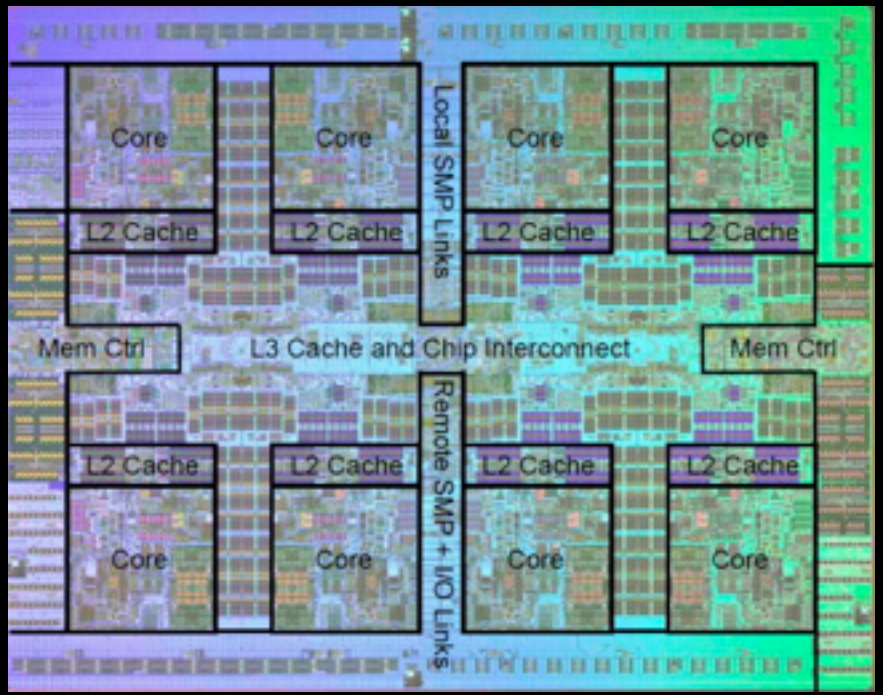
HTC (and many other smartphone companies) buy chips from Qualcomm.

Fabless Captive Model

Apple designs chips (the "A" series) for exclusive use in its iOS-series products (such as the iPhone).

Apple doesn't own a fab, and so it contracts with foundries (currently Samsung!)

Original Captive Model



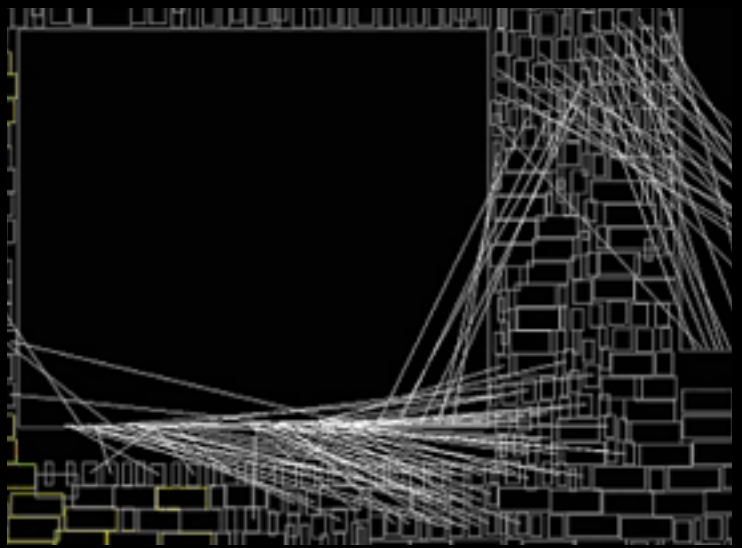
IBM owns fabs, and designs CPU chips, for use in most of its server lines.

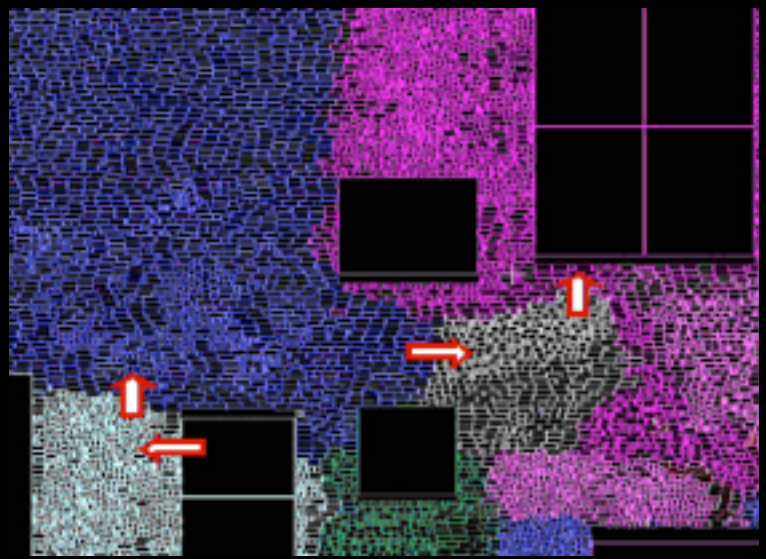
For many decades, it manufactured DRAM chips (Robert Dennard, the inventor of the DRAM, works for IBM).

Entire industries exist to "sell arms" to "semis"

"Electronic Design Automation" (EDA, or CAD)

SYNOPSYS®
Accelerating Innovation





Chip fabrication tools

KLA-Tencor
Accelerating Yield

Wafer manufacturing

Silicate materials form 90% of the earth's crust

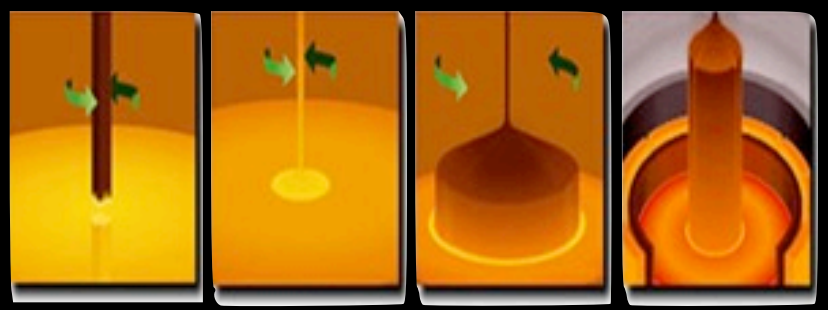
Silica (silicon dioxide) is mined, mostly for use in concrete.

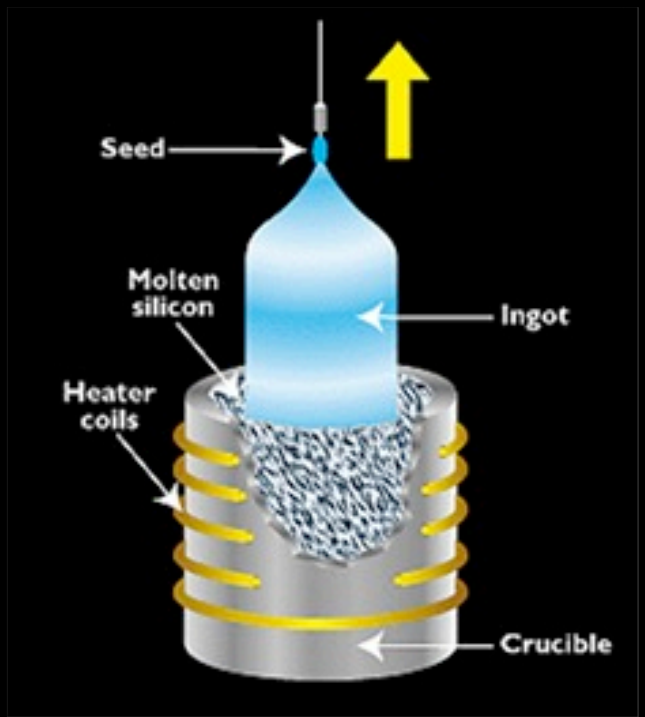
Refined elemental silicon (95% pure) is mostly used for aluminum metallurgy.

Electronic grade silicon requires 99.9999999% purity.

"Single crystalline" composition is also required for wafers.

Silicon "ingots" are grown from a "perfect" crystal seed in a melt, and then purified to "nine nines".





Ingots sliced into 450μm thick wafers, using a diamond saw.

This photo is from Intel's ecommerce site for server chips.

This photo is from Intel's ecommerce site for server chips.

6 inch square Si single-crystal wafer, $2 on Alibaba

Wafer costs are crucial for solar cells, but irrelevant for chips that perform high-value functions.

Xeon E7-8870, in small quantities, $4600

But yet, photo was shot behind solar cells to look "hi-tech" :-).

Fabrication

On Thursday

Advanced pipeline design - welcome to the 1990s !

Th 2/6	Super-Pipelining + Branch Prediction		Appendix C.2,6, Chapter 3.3.

Have fun in section!