# CS 152
# Computer Architecture and Engineering
# Lecture 20 -- Dynamic Scheduling III

**2014-4-8**

**John Lazzaro**

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

TA: Eric Love

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

Play:

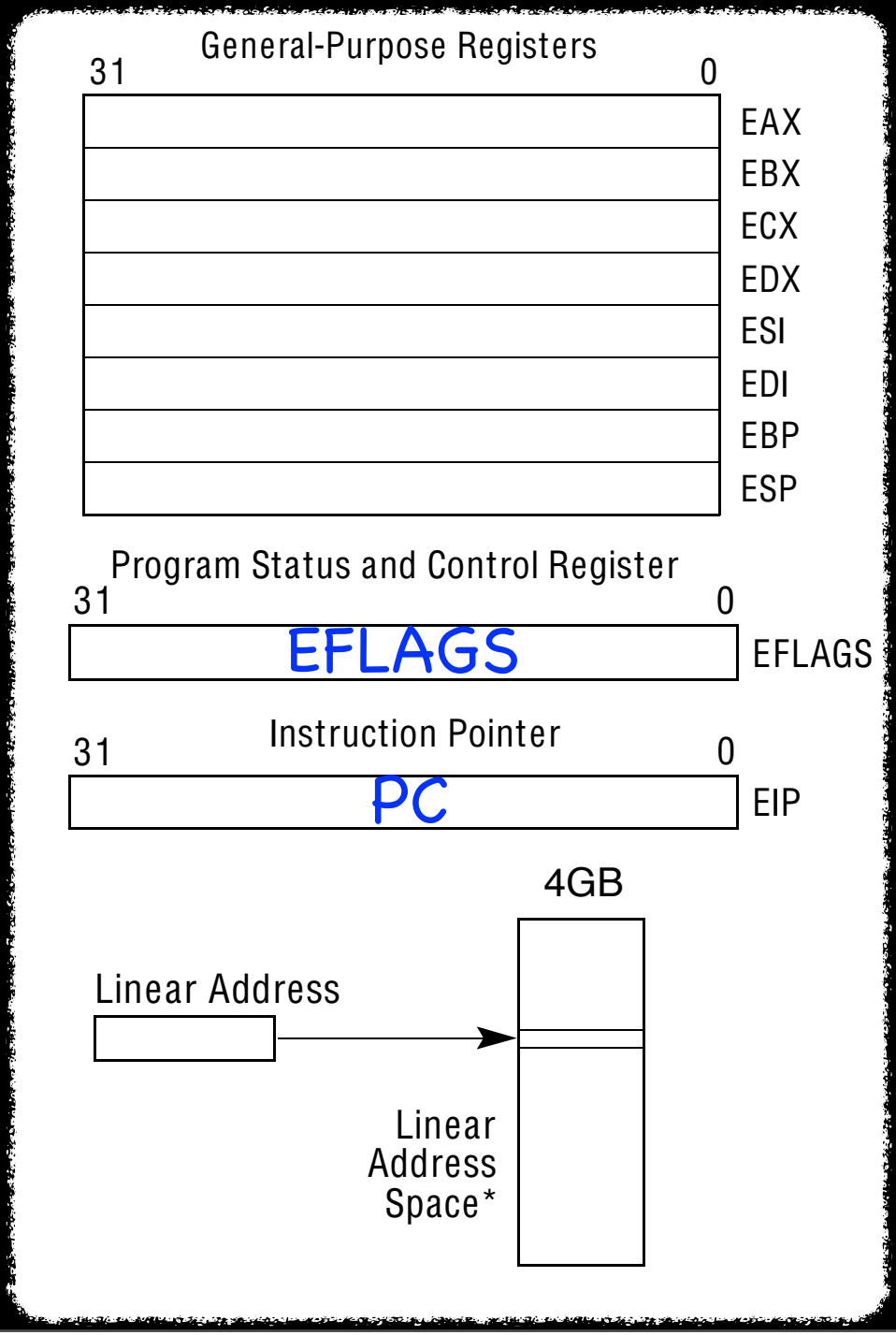
# Intel Inside: Dynamic Execution

- **The IA-32 ISA:** How to adapt a complex instruction set to dynamic techniques.
- **Pentium IV:** Case study of the Netburst micro-architecture, introduced in 2001.

## Short Break

- **Sandy Bridge:** Dynamic execution in today's Core and Xeon product lines.
- **Limits to Parallelism:** How fast can the **perfect** dynamic execution design go?

# IA-32, optimist view




When configured to conceal "legacy" modes ...

Two differences between MIPS and IA-32 integer programmer's model:

- Only 8 registers
  Thus, register renaming.
- ALU instructions set bits in EFLAGS register. Branches are taken based on EFLAGS values.

Example:

Zero Flag (ZF): ZF = 1 if last ALU instruction result was zero, otherwise ZF = 0.

# IA-32, instruction complexity

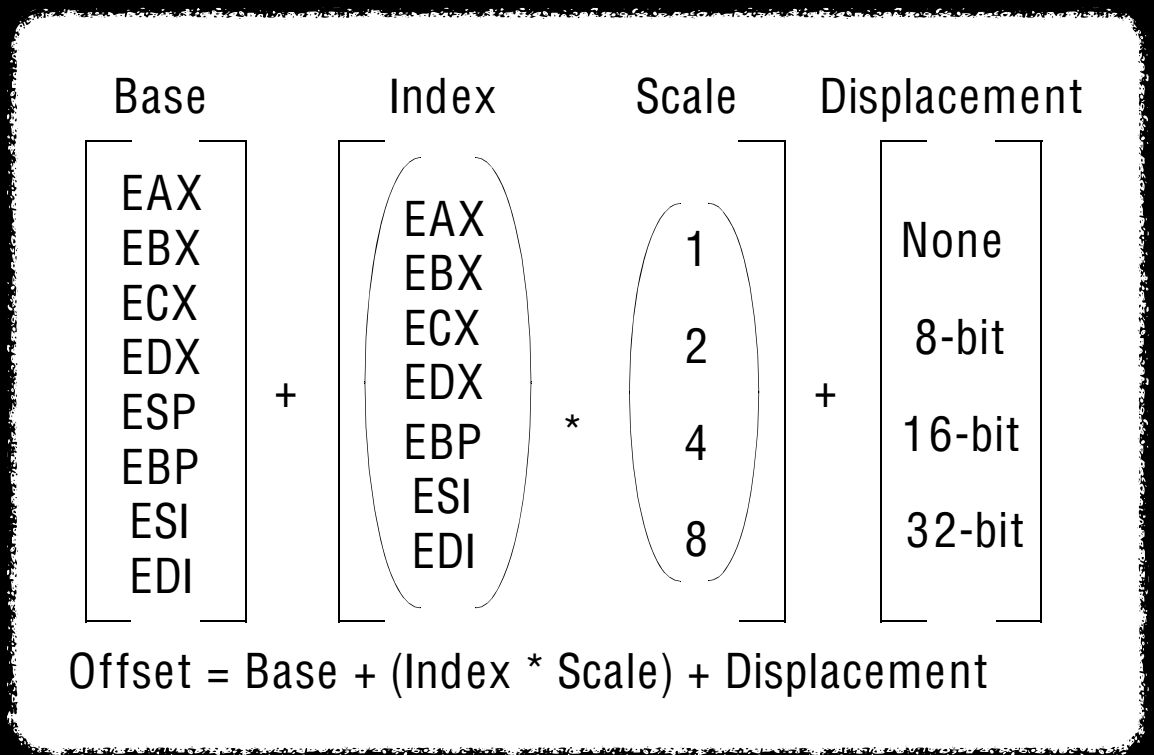
Even the simple instructions are complex:

ADC m32, r32:

Read the memory location m32, add the contents of register r32 to it, along with the carry bit in EFLAGS, and store the result back in m32.

Rewriting this instruction in MIPS takes at least four instructions: 1 load, 2 adds, 1 store. More for a complex m32 addressing mode.

# IA-32, addressing modes

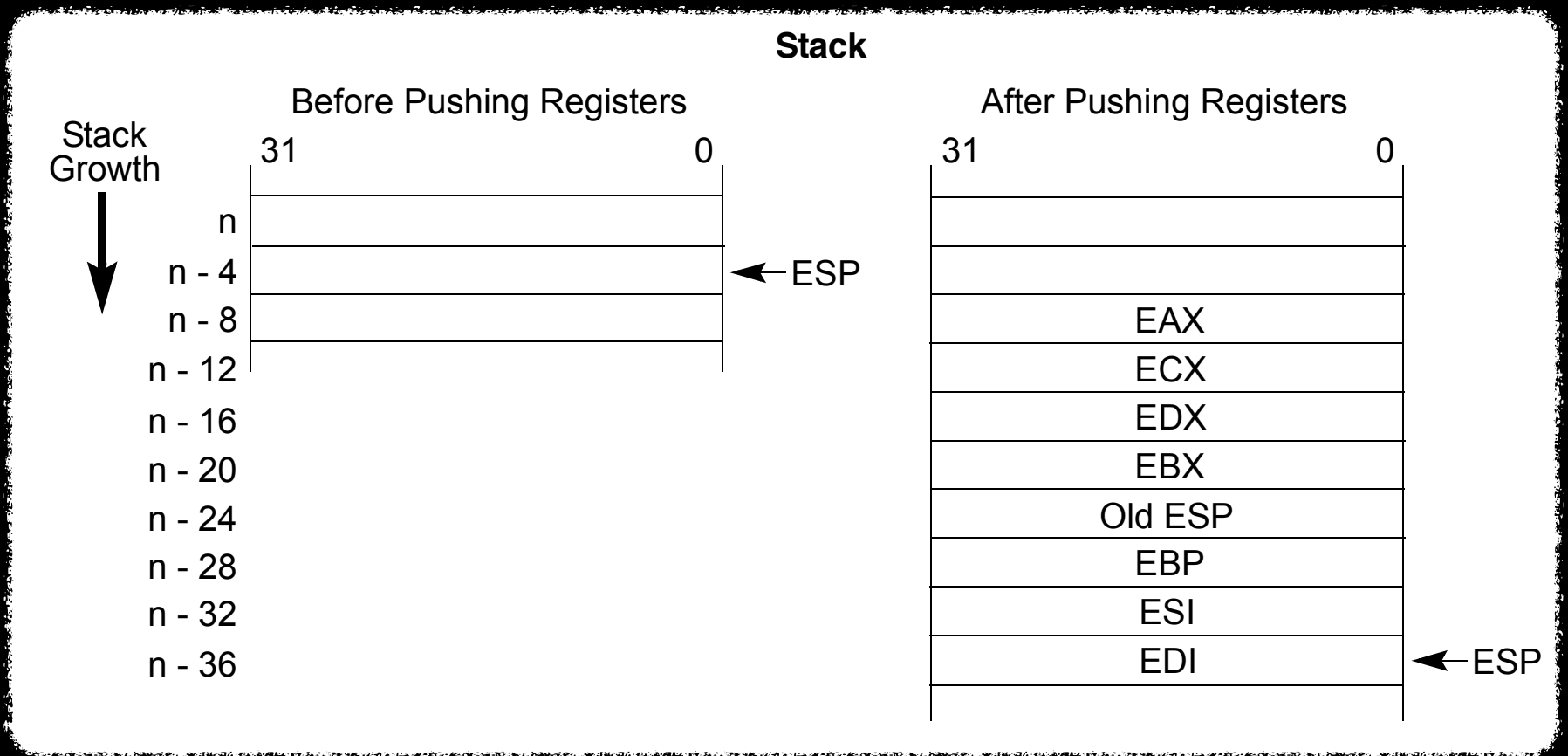



Up to 2 adds and a shift for a memory operand …

# IA-32, instruction complexity

And then there are complex instructions …

PUSHA: Push all 8 integer registers onto the stack




Rewrite PUSHA in MIPS? > 16 instructions.

Some IA-32 instructions require 100s of MIPS instructions …

# IA-32, instruction decode

IA-32 instructions are 1 to 17 bytes in length.

To determine an instruction's length? Parse it!

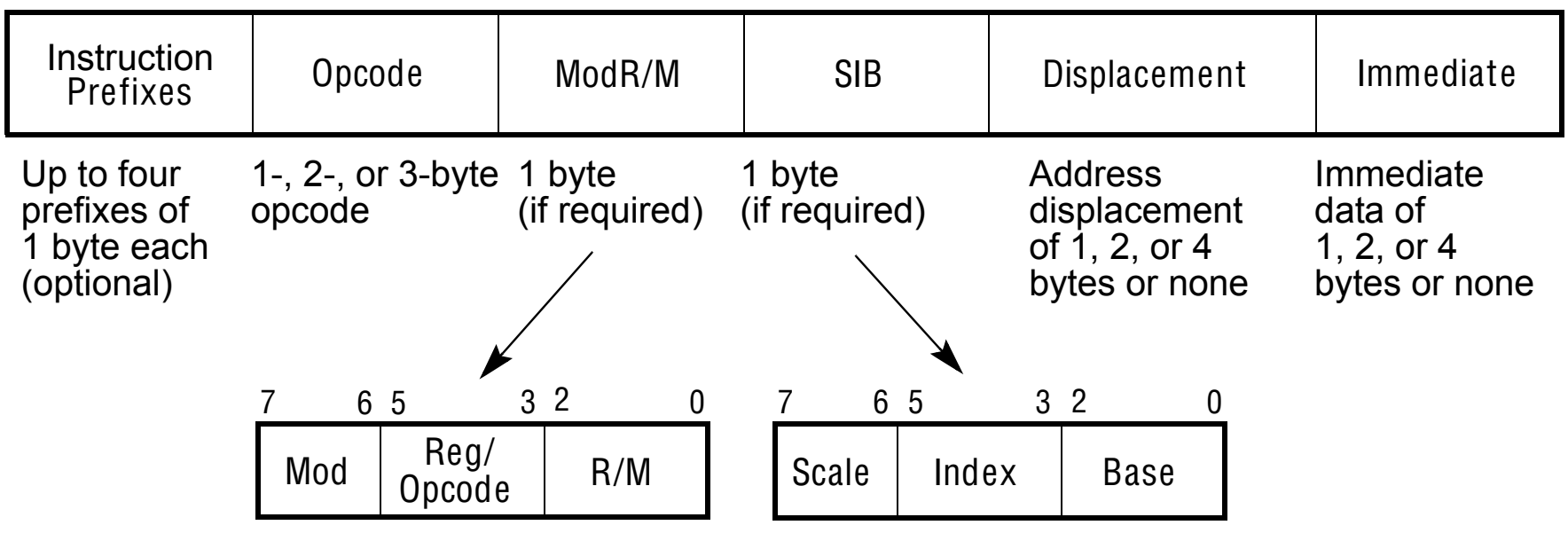



Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

Creeping features: Prefix bytes for specifying atomicity, for hinting branch direction, for 8086 mode ...

# Dynamic Execution?

# Idea: Translate IA-32 opcodes to RISC

Micro-ops:

RISC-like instructions that reference the 8 architected registers, plus "temporary" registers.

Translate?

For IA-32 instructions that map into a small number of micro-ops, do translation after decode.

EFLAGS?

Treat each EFLAG bit as an architected register.

# Translation example ...

ADC m32, r32: // for a simple m32 address mode

Becomes:

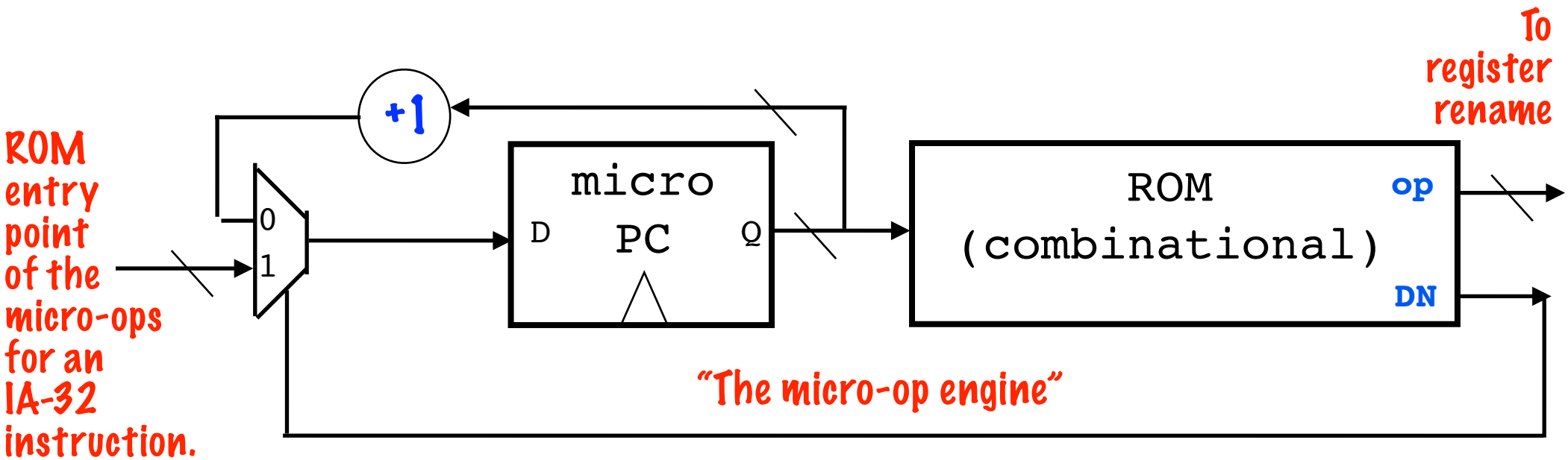
```
LD T1 0(EBX);   // EBX register point to m32
ADD T1, T1, CF; // CF is carry flag from EFLAGS
ADD T1, T1, r32; // Add the specified register
ST 0(EBX) T1;   // Store result back to m32
```

Instruction traces of IA-32 programs show most executed instructions require 4 or fewer micro-ops.

Translation for these ops are cast into logic gates, often over several pipeline cycles.

# Complex instructions? Microcode ROM

Instructions like PUSHA are sent to a sequencer that clocks a long sequence of micro-ops out of a ROM.



# Power-hungry decode? Cache micro-ops!

Most IA-32 implementations cache decoded instructions, to avoid repetitive decoding of the complicated instruction format.

Implementations vary in the size of the caches, and how the micro-op cache interacts with the branch predictor and the normal instruction cache.

Macro-Fusion: Another idea to save power, by reducing the number of micro-ops in a translation. An implementation defines micro-ops that captures the semantics of IA-32 instruction pairs that occur in sequence, and maps these pairs to one micro-op.

Intel Pentium IV (2001)

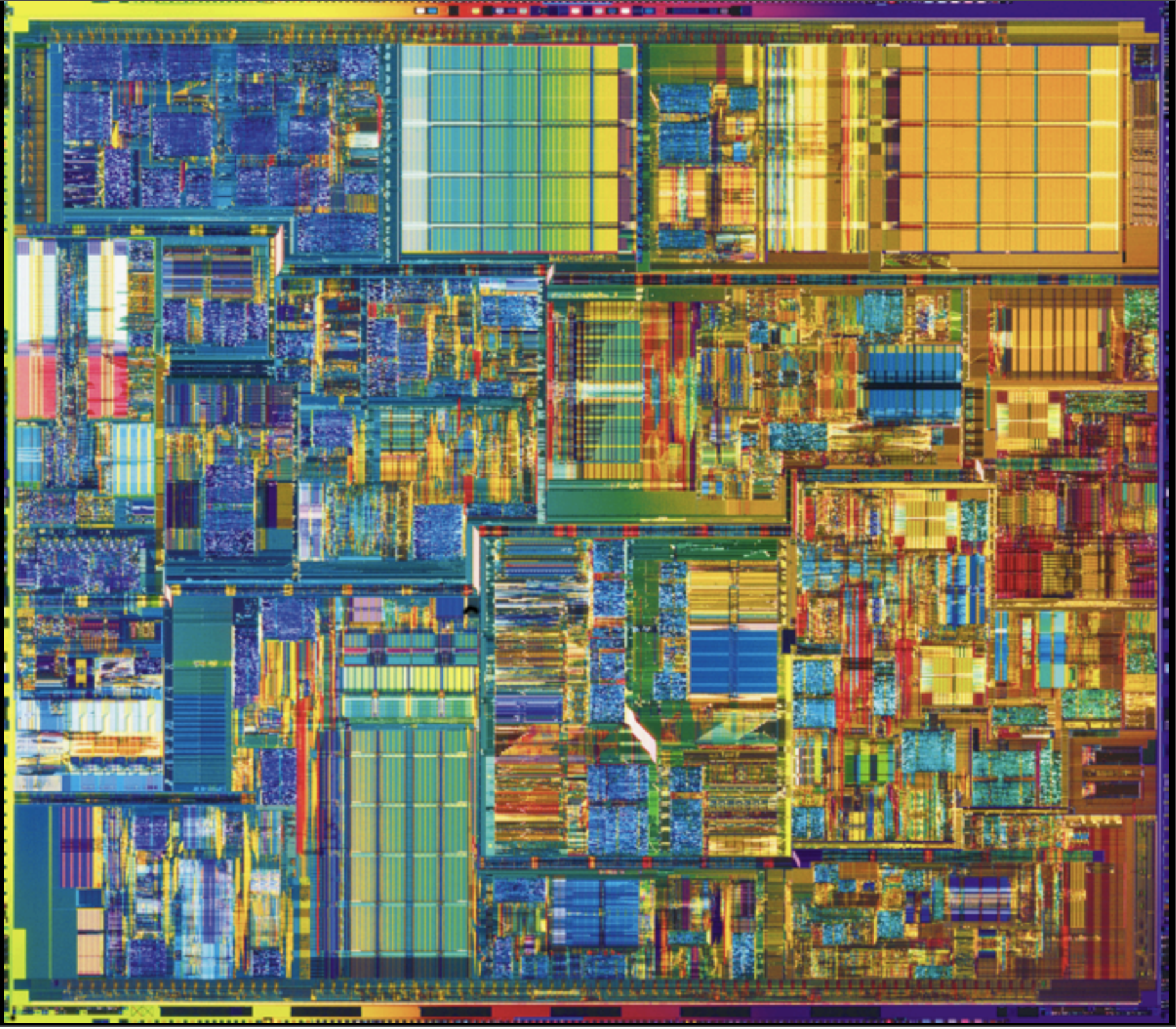
1.5 GHz main clock

Execute loop has 2X clock

55 Watts

0.15um process

42M transistors

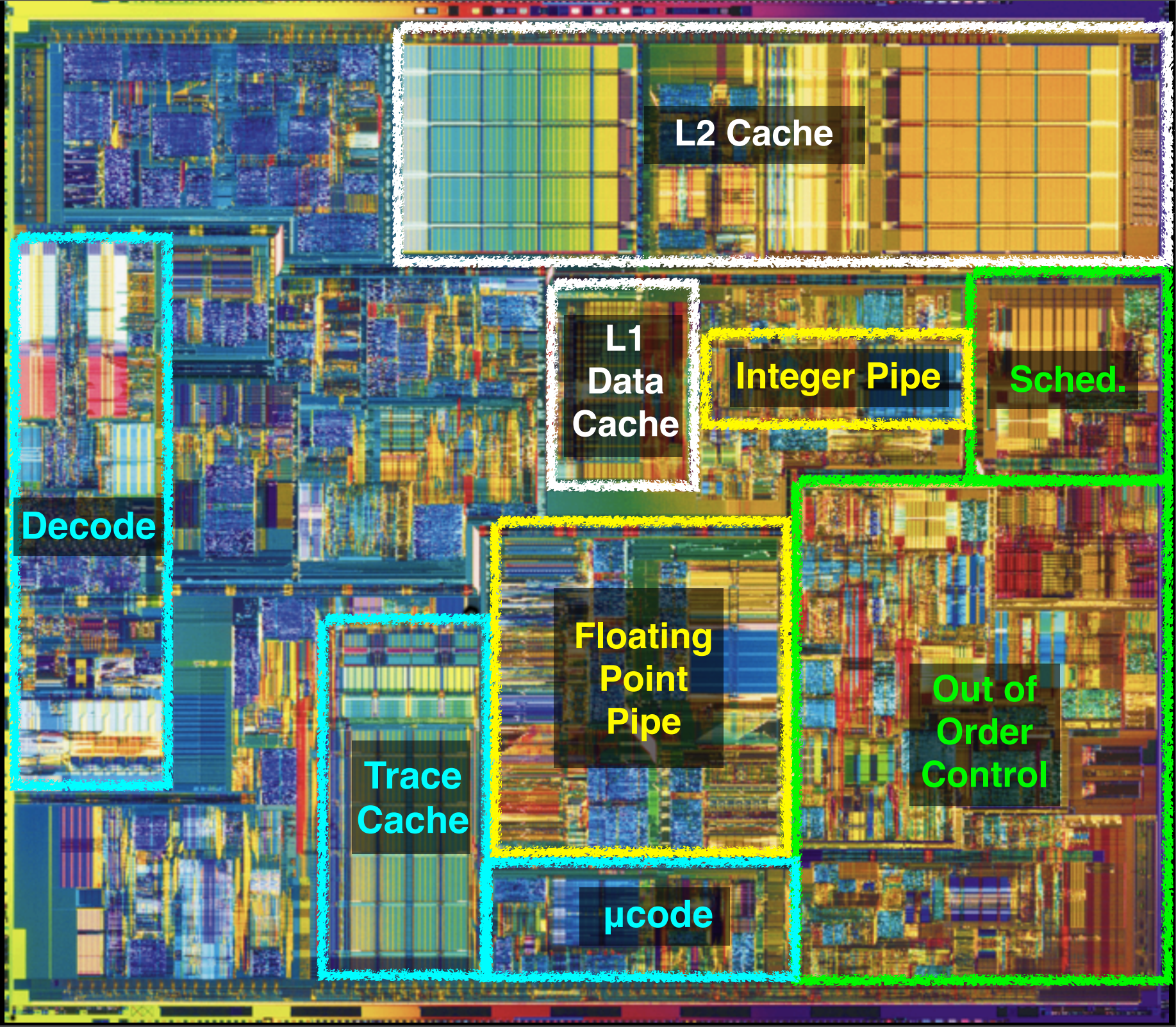


Intel Pentium IV (2001)

Cyan blocks "fix" IA-32 issues

Green blocks are for OoO

Gold blocks are fast execute pipes

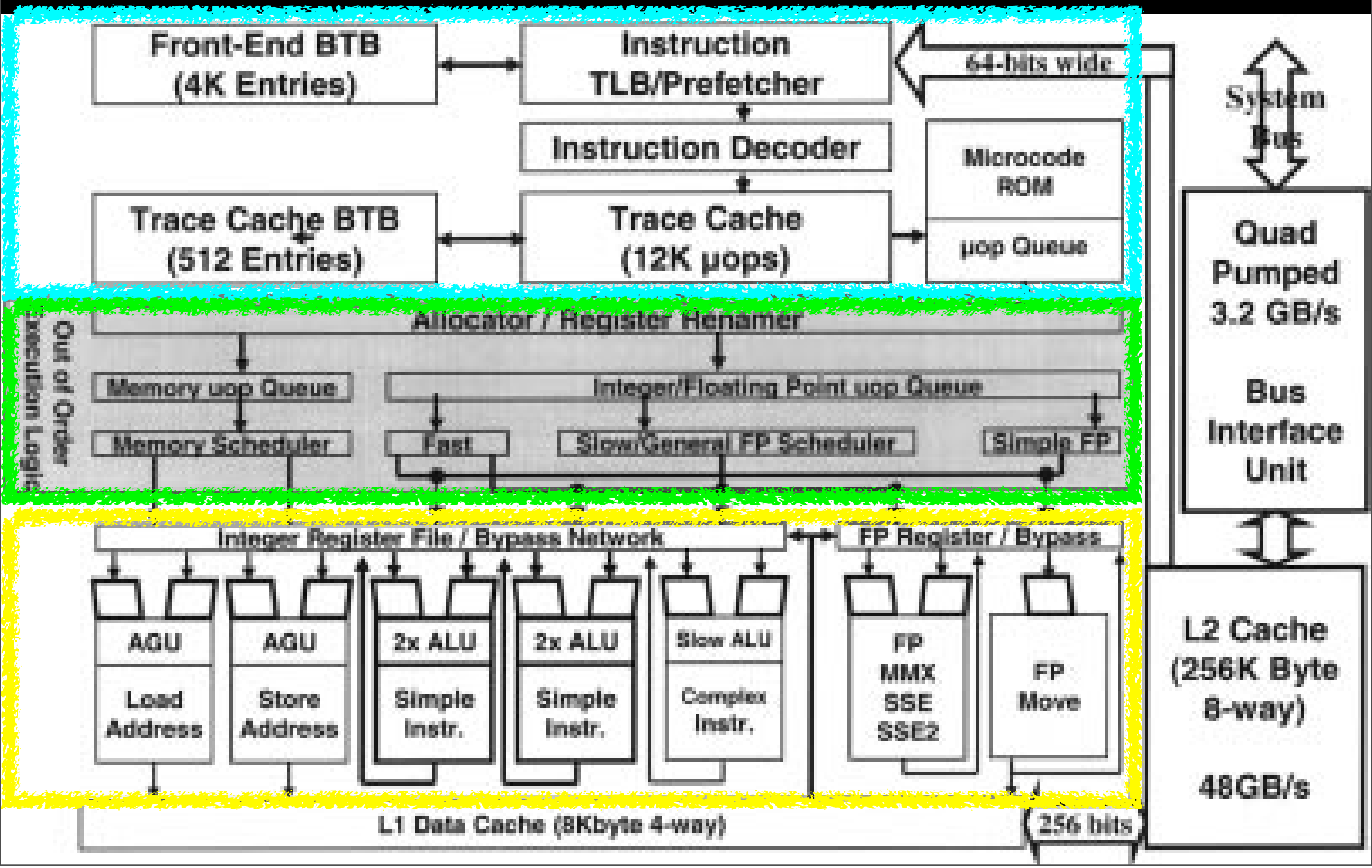


Intel Pentium IV (2001)

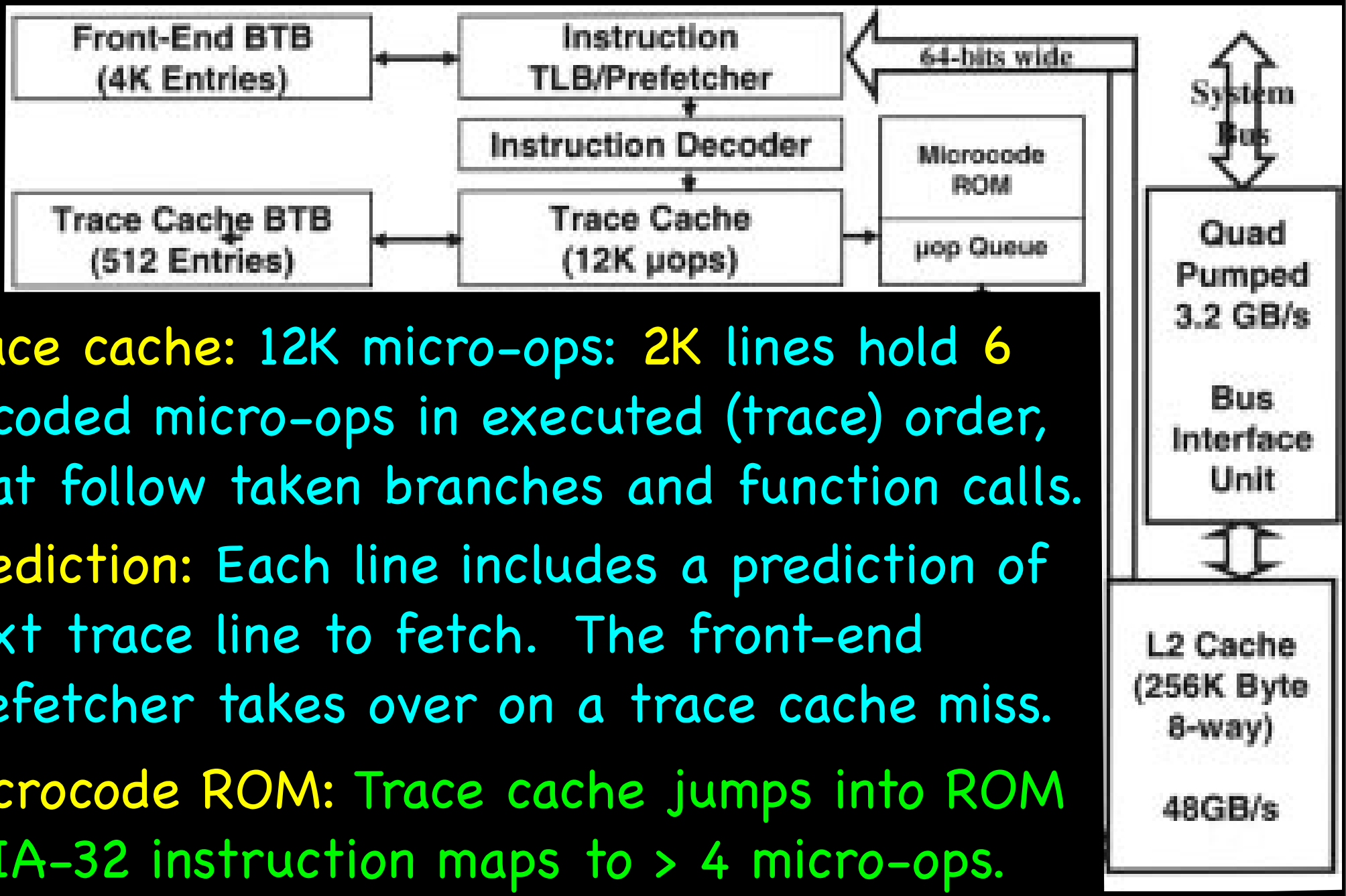
Cyan blocks "fix" IA-32 issues

Green blocks are for OoO

Gold blocks are fast execute pipes



Branch predictor steers instruction prefetch from L2.
Maximum decoder rate is 1 IA-32 instruction per cycle.

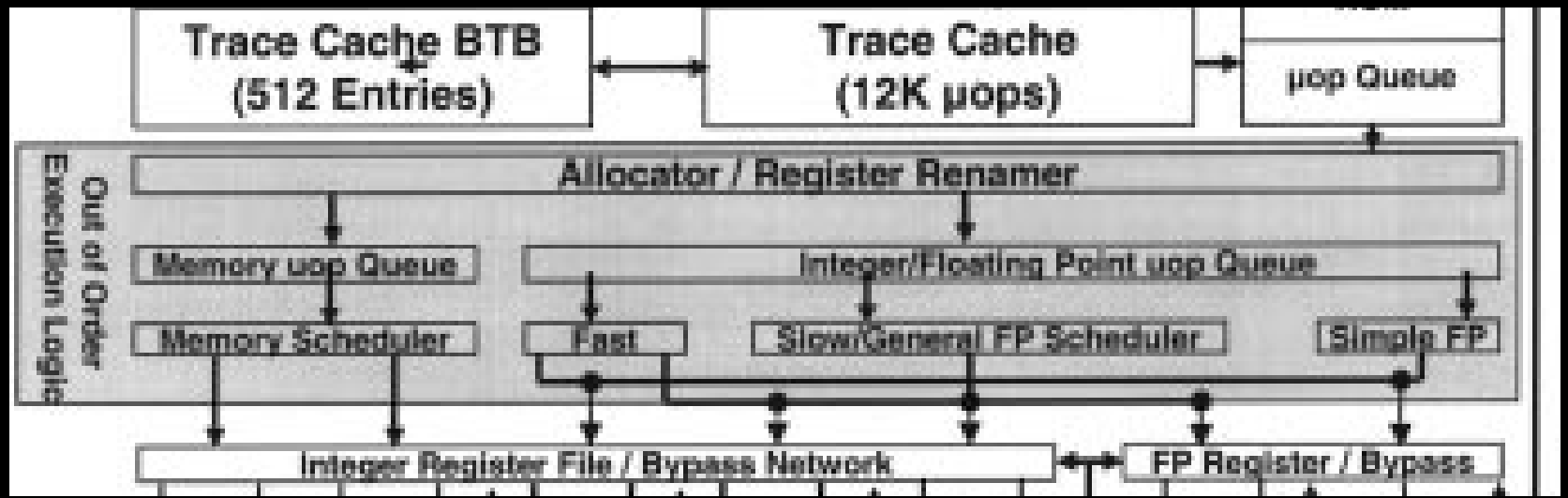



Trace cache: 12K micro-ops: 2K lines hold 6 decoded micro-ops in executed (trace) order, that follow taken branches and function calls.

Prediction: Each line includes a prediction of next trace line to fetch. The front-end prefetcher takes over on a trace cache miss.

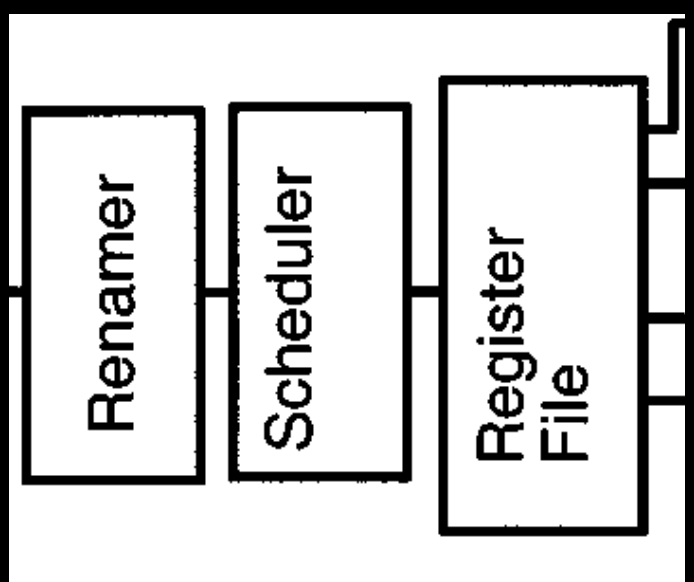
Microcode ROM: Trace cache jumps into ROM if IA-32 instruction maps to > 4 micro-ops.

Out of order logic




126 instructions in flight

3 micro-ops per cycle from trace cache

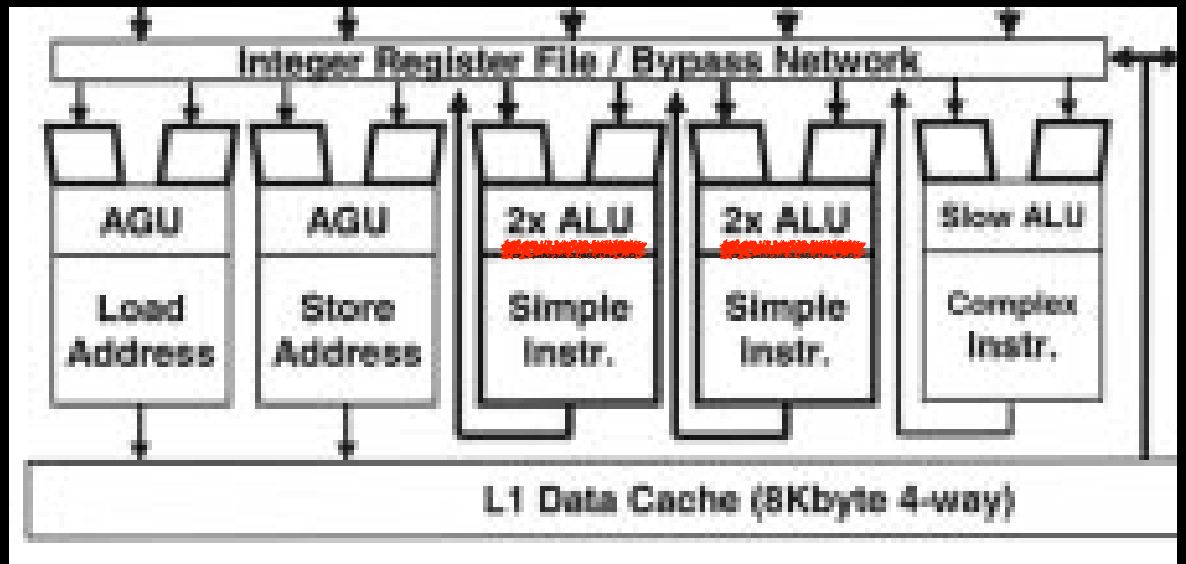



6 operations scheduled per cycle

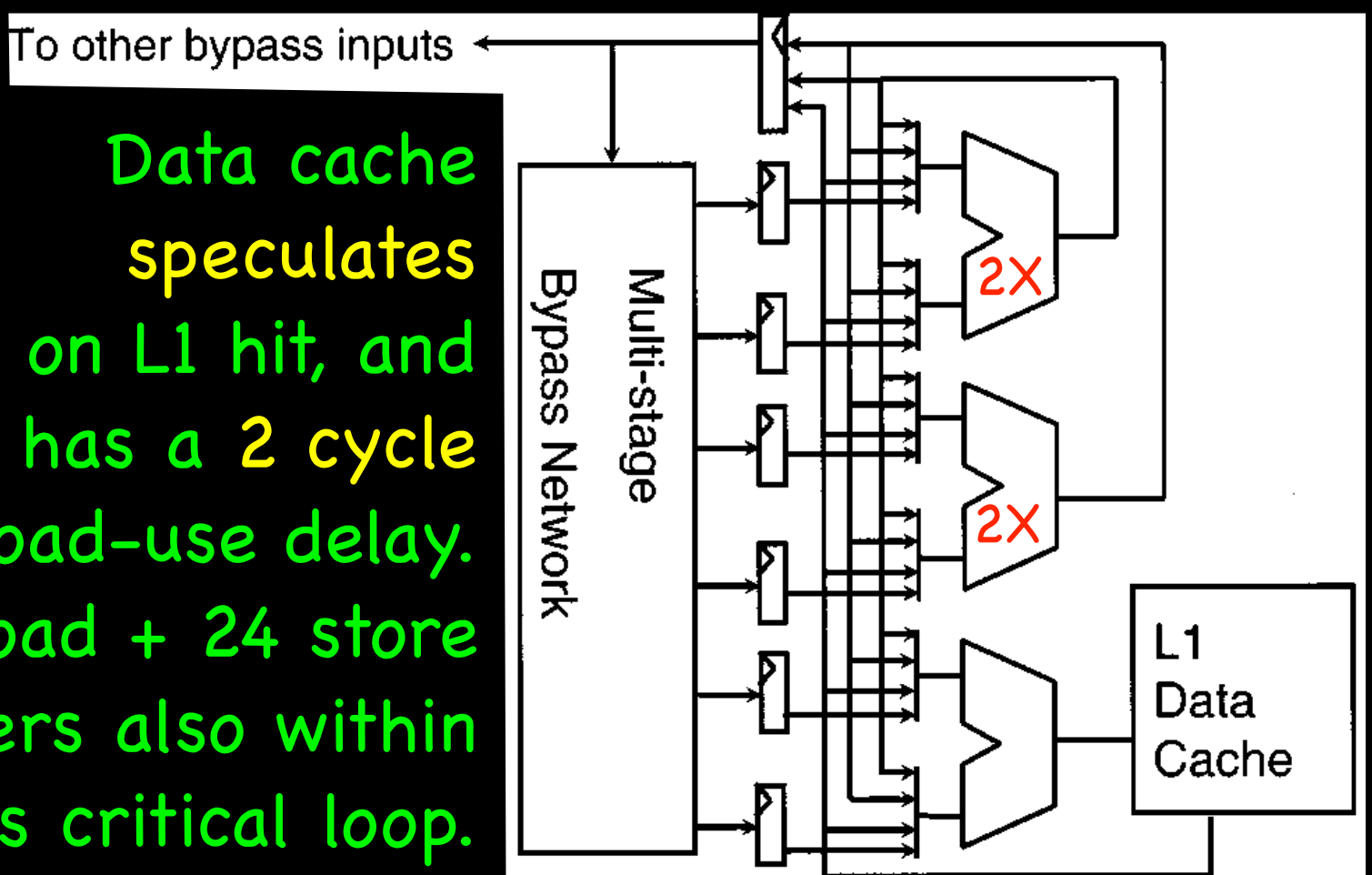
3 micro-ops retired per cycle

128 integer physical registers
128 floating point physical registers

Execution Unit:
Simple integer ALUs runs on both edges of the 1.5 Ghz clock.

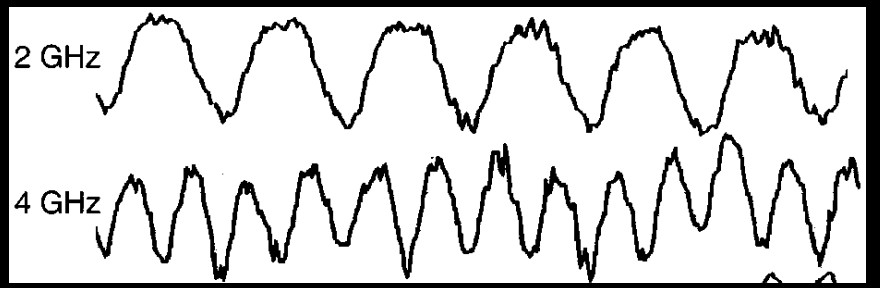



Data cache speculates on L1 hit, and has a 2 cycle load-use delay. 48 load + 24 store buffers also within this critical loop.

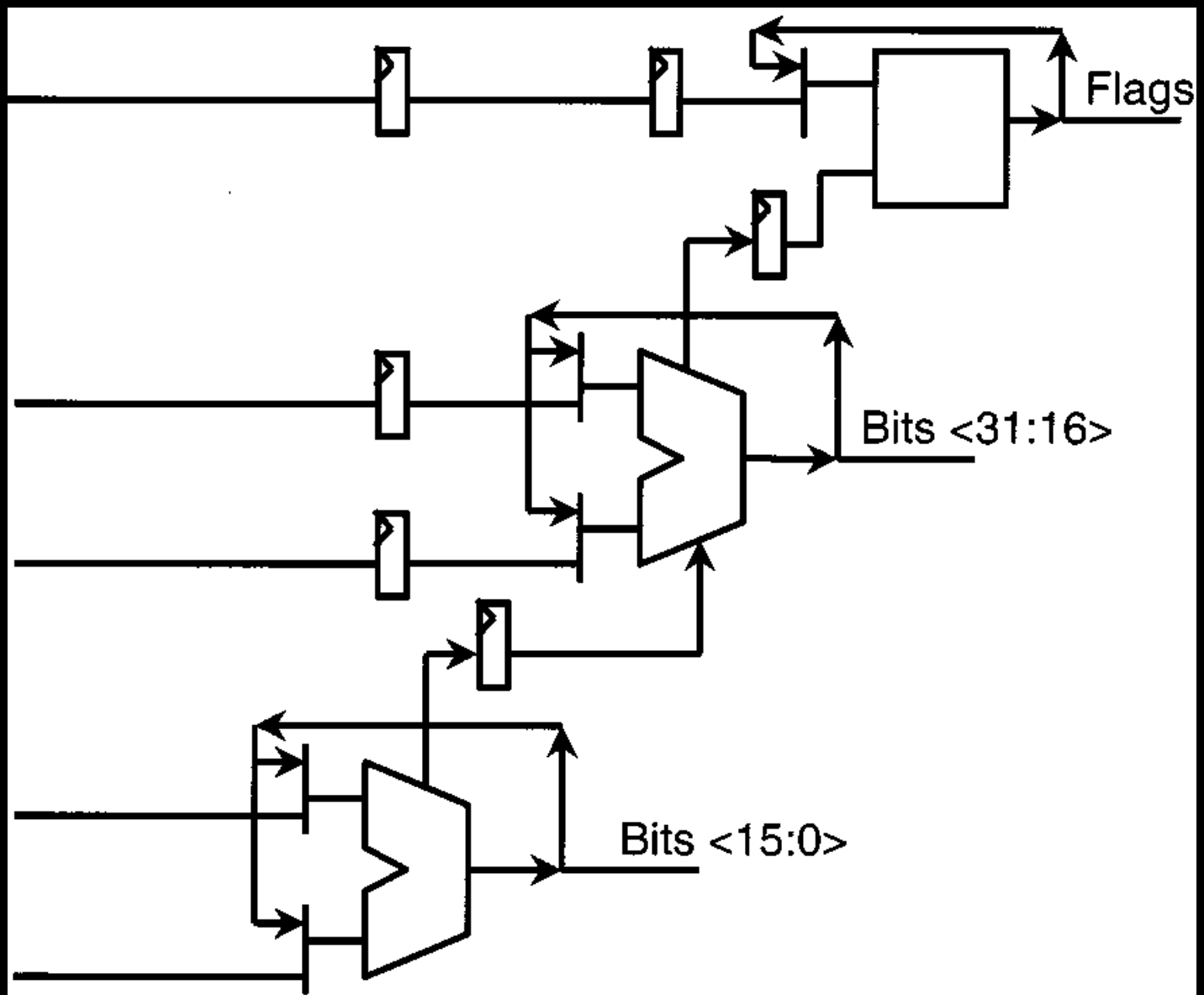



2X Bypass Network: Fast ALUs can use their own results on each edge.

# Key trick: Staggered ALUs

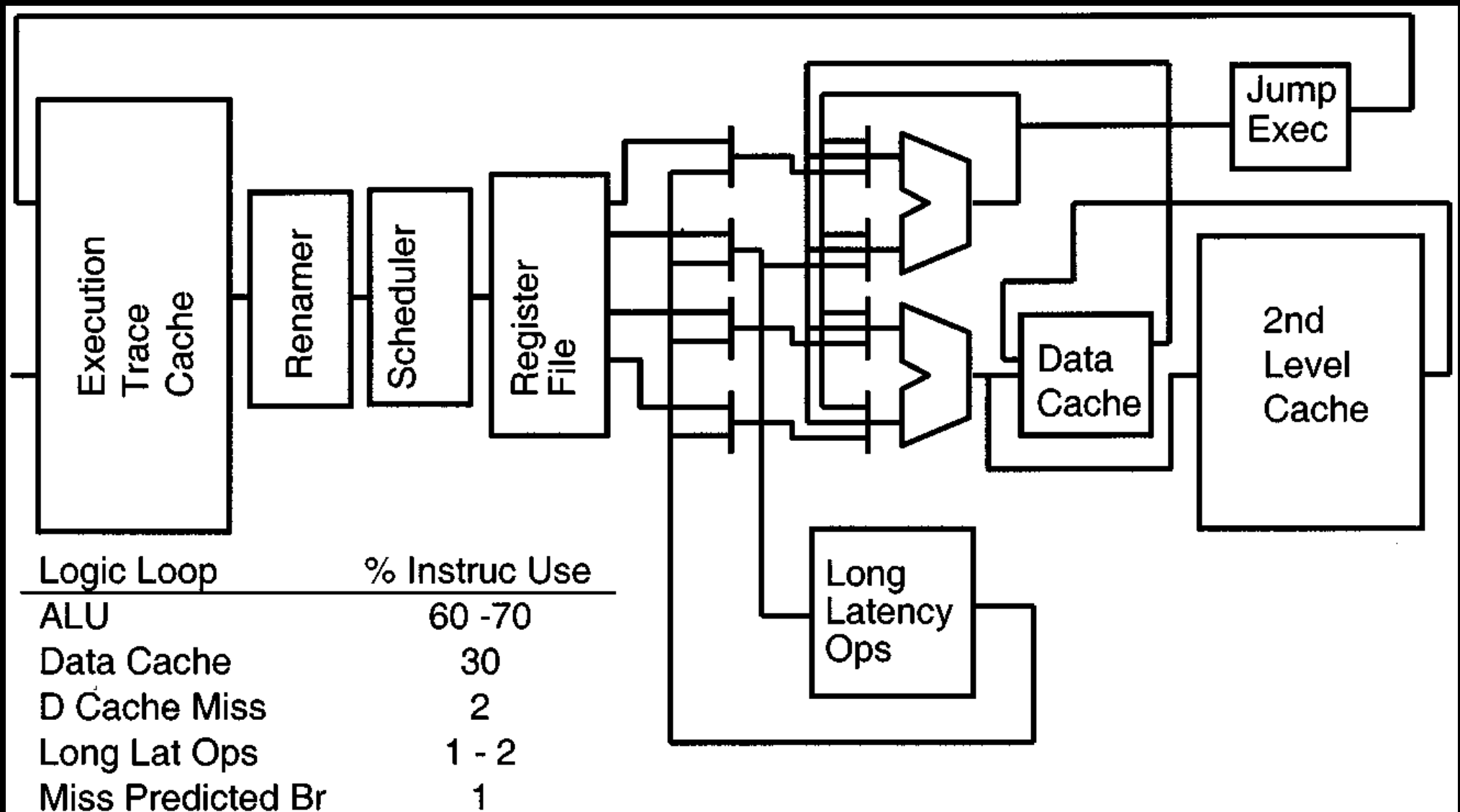



Pipeline registers added to carry chain. A 32-bit adder is computing parts of 3 different operations at the same time.



# In context: complete datapath

The logic loops used 90% of the time run at 3 GHz, but most of the chip runs at 1500 MHz.




| Logic Loop | % Instruc Use |
|---|---|
| ALU | 60 -70 |
| Data Cache | 30 |
| D Cache Miss | 2 |
| Long Lat Ops | 1 - 2 |
| Miss Predicted Br | 1 |

# In actuality: The clock went too fast

## Basic Pentium III Misprediction Pipeline

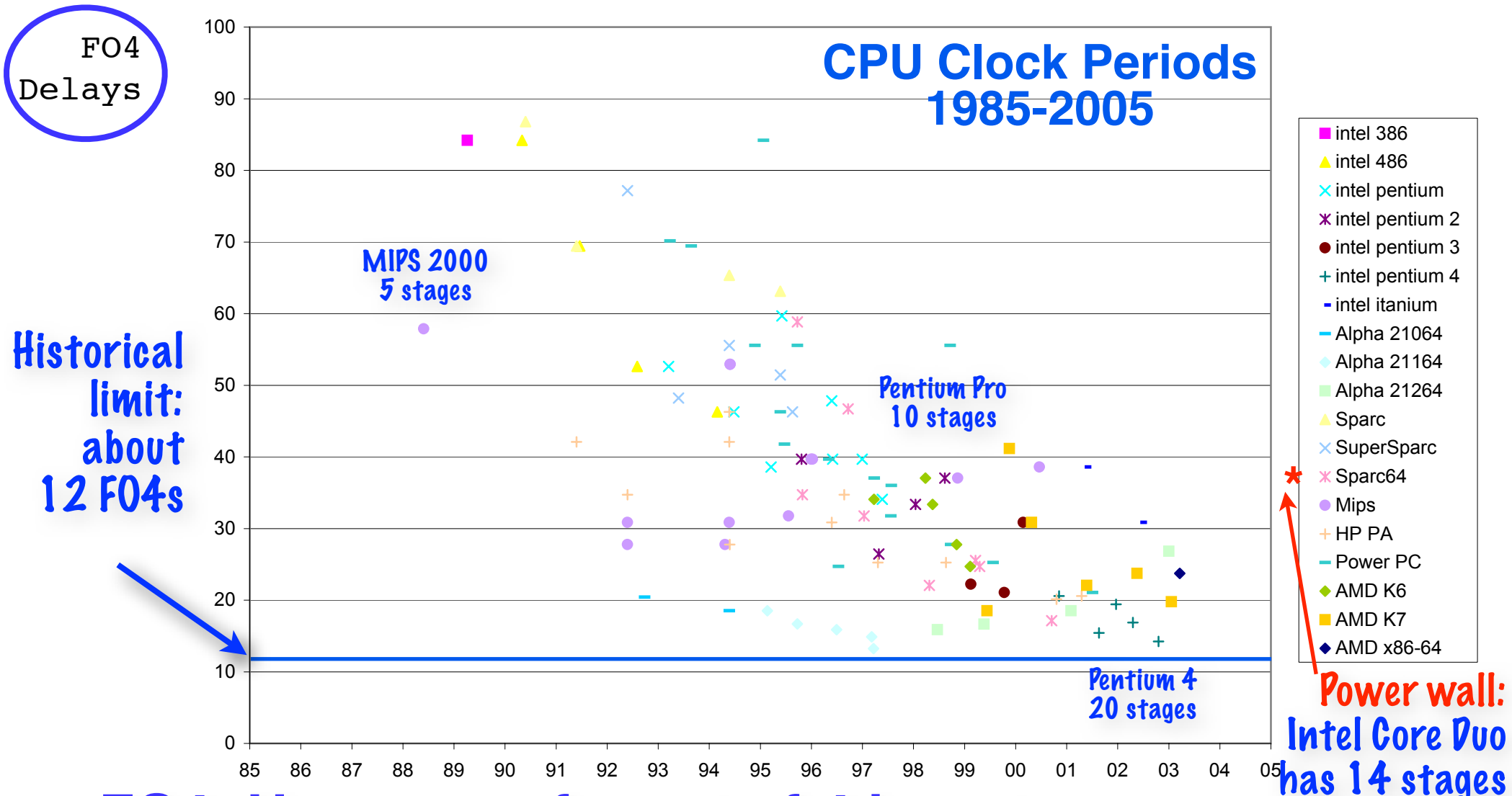
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| Fetch | Fetch | Decode | Decode | Decode | Rename | ROB Rd | Rdy/Sch | Dispatch | Exec |

## Basic Pentium 4 Misprediction Pipeline

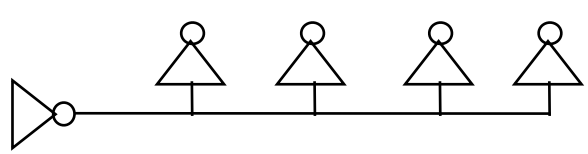
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TC Nxt IP | | TC Fetch | | Drive | Alloc | Rename | | Que | Sch | Sch | Sch | Disp | Disp | RF | RF | Ex | Figs | Br Ck | Drive |

The trace cache was too much of a departure from Pentium III, and the existing code base missed the cache too often. This was particularly bad because the Pentium IV pipeline had so many stages!

# Recall: Limits to super-pipelining ...




**FO4: How many fanout-of-4 inverter delays in the clock period.**



Thanks to Francois Labonte, Stanford

The Pentium IV was the chip that foreshadowed the "power wall".

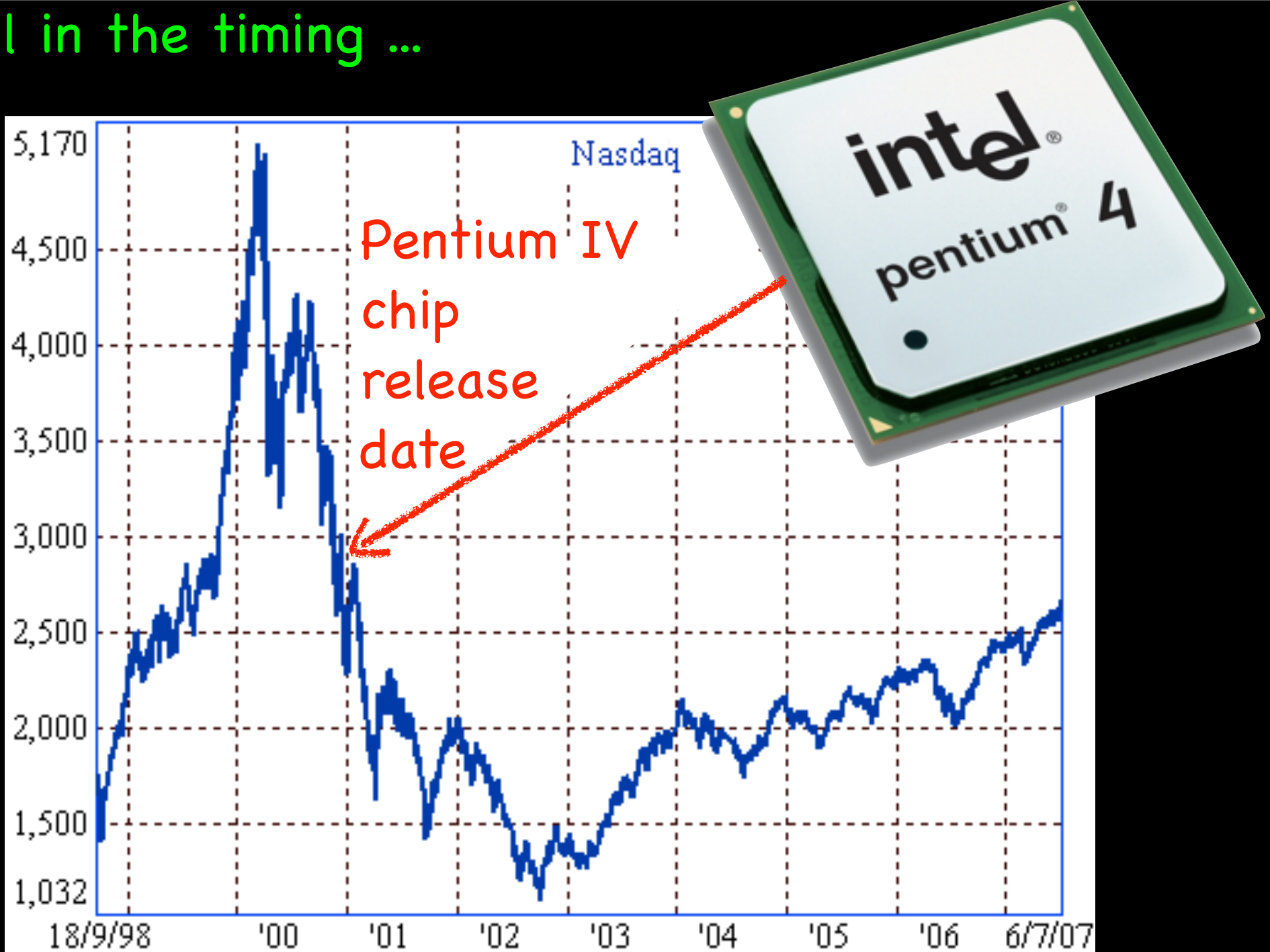
Upper-management pressure for a high clock rate (for marketing) pushed the design team to use too many pipeline stages, and performance (and power) suffered.


---

Intel recovered by going back to their earlier Pentium Pro out-of-order design ...

Many elements of the Pentium IV are innovative ... and were reintroduced in Sandy Bridge (2011-onward).

# It's all in the timing ...

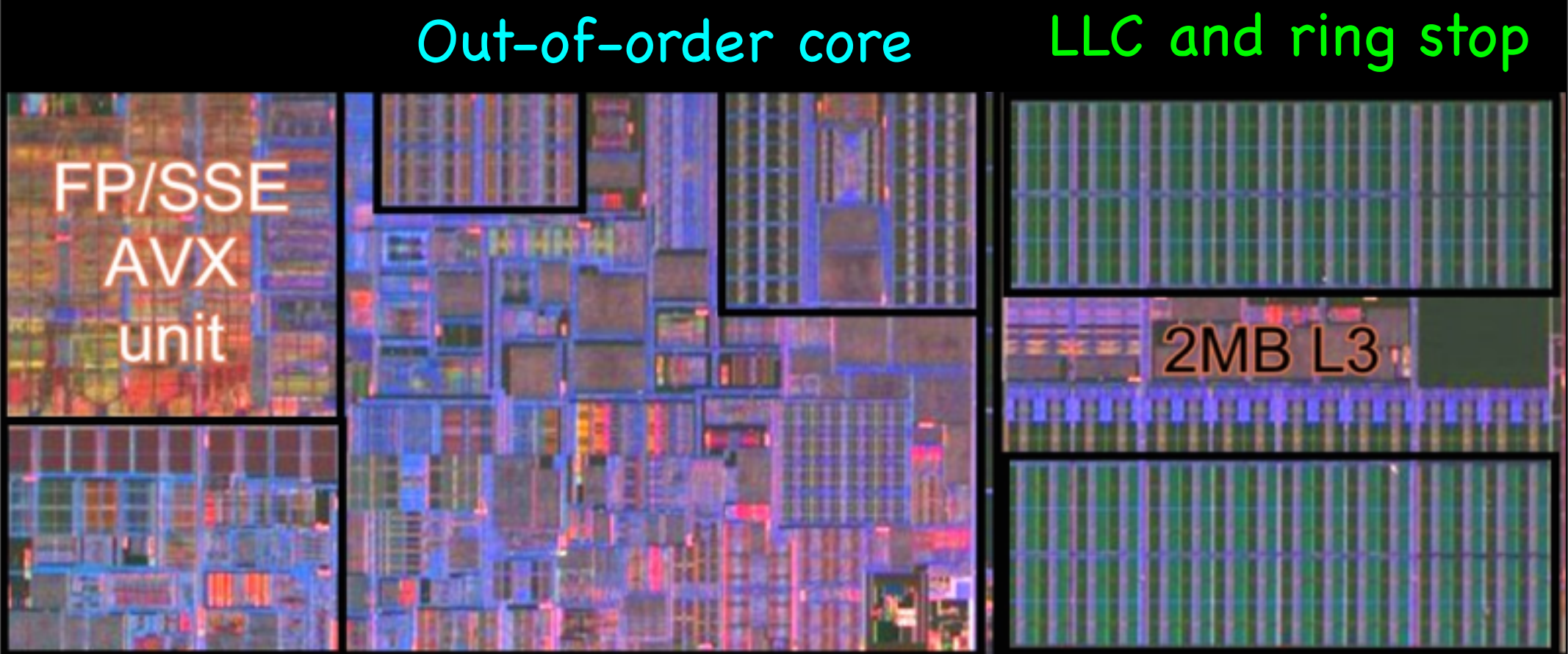



The dot-com tech stock "bubble"

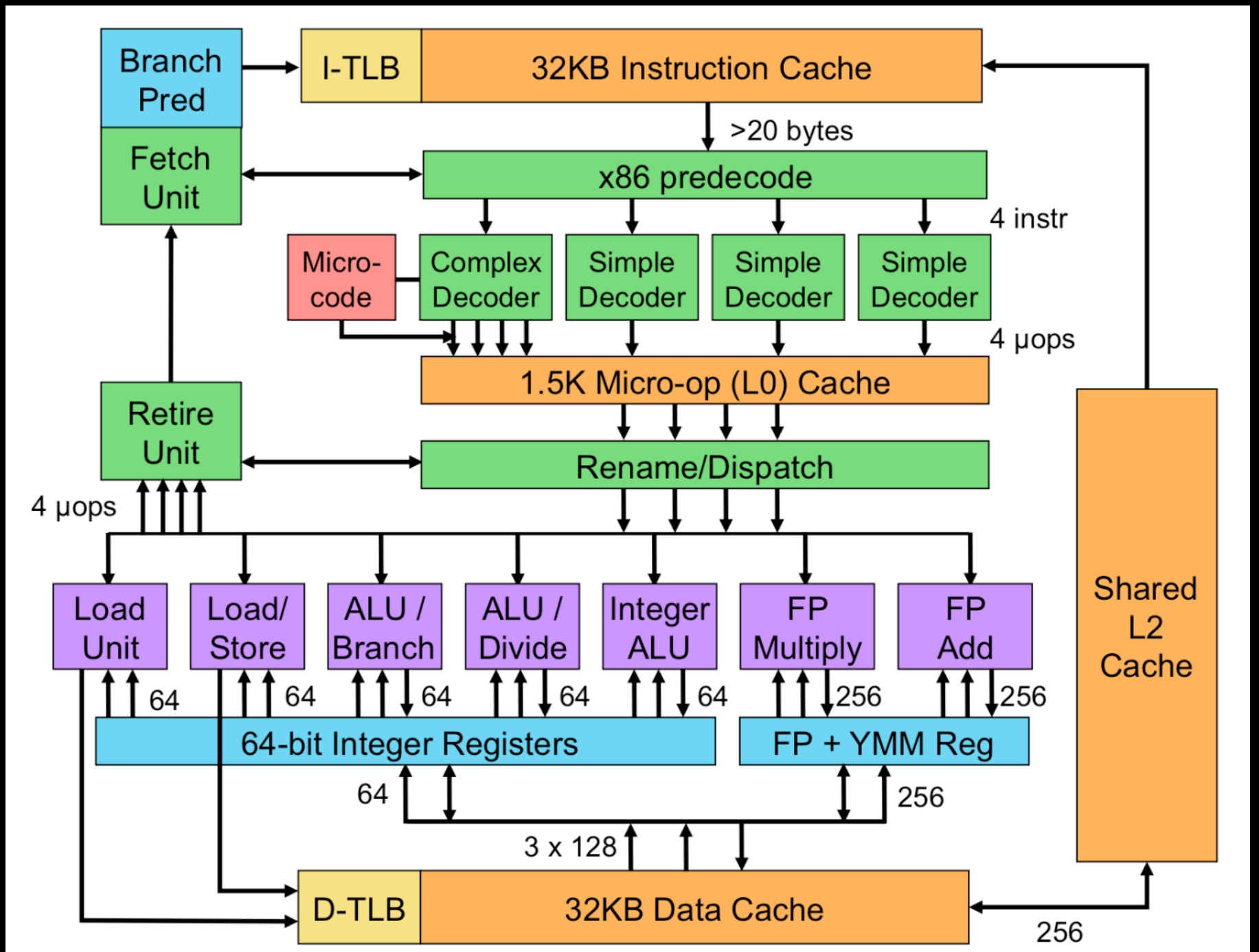
# Break

Play:

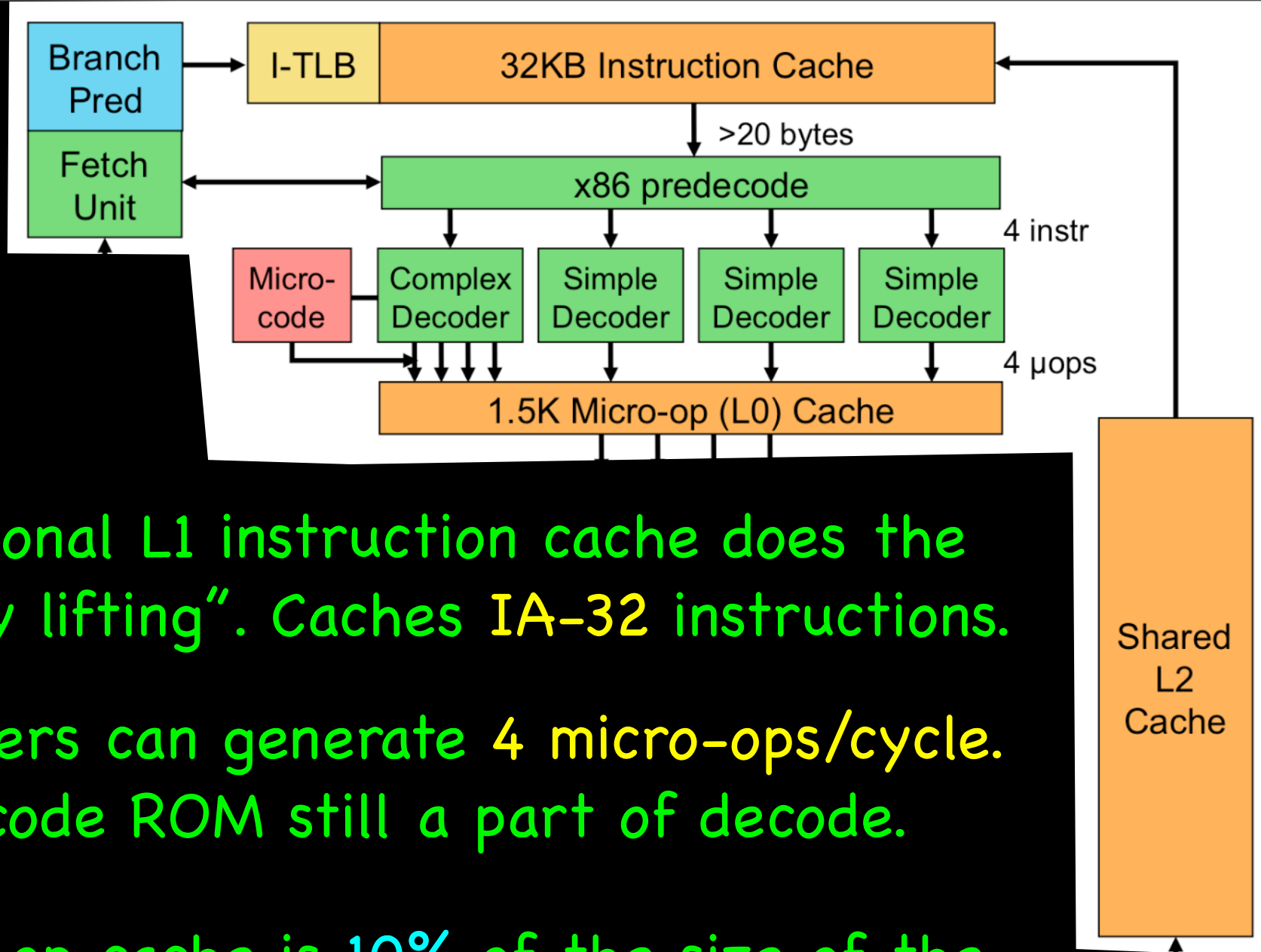
# Sandy Bridge



# Sandy Bridge

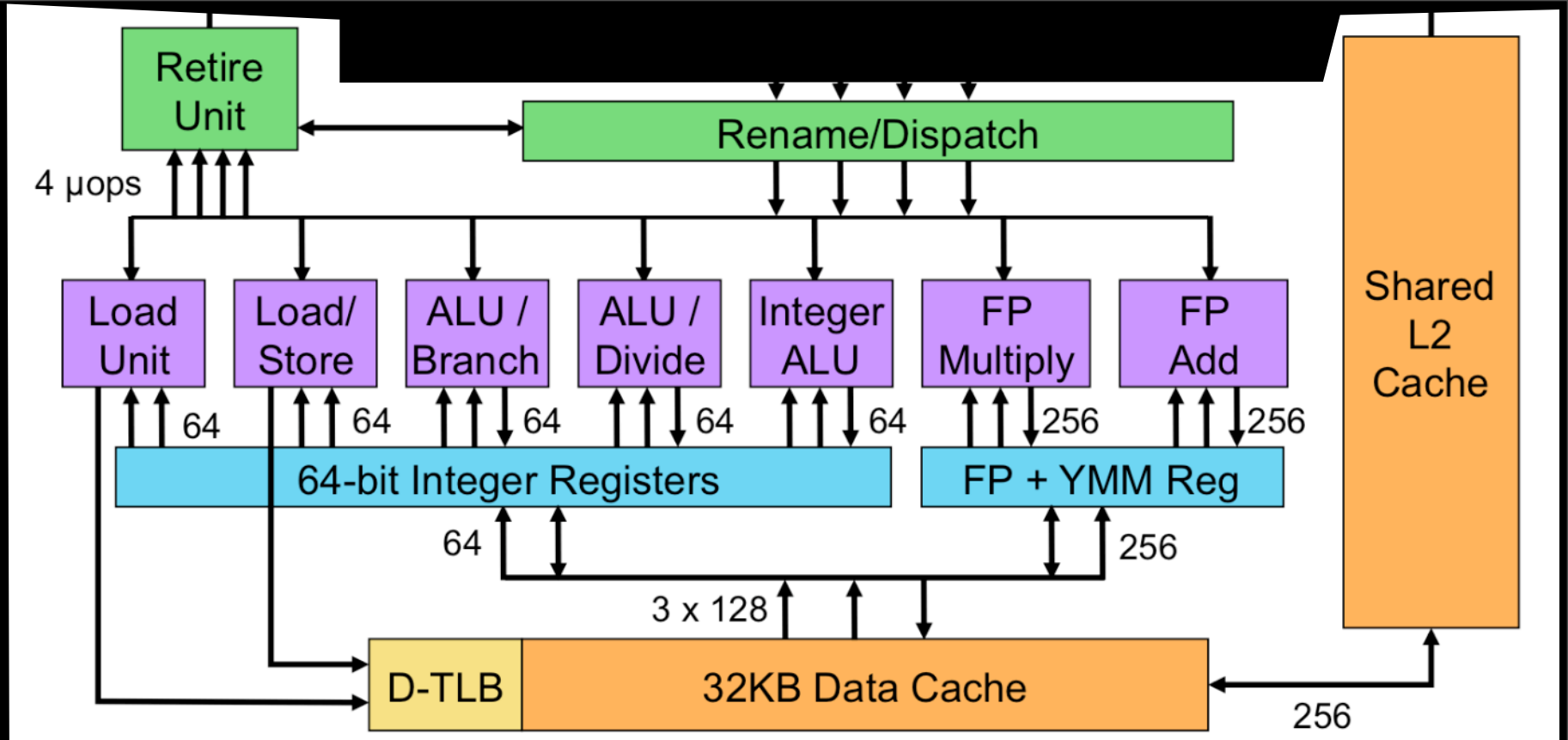


# Front End




- Traditional L1 instruction cache does the "heavy lifting". Caches IA-32 instructions.
- Decoders can generate 4 micro-ops/cycle. Microcode ROM still a part of decode.
- Micro-op cache is 10% of the size of the Pentium IV trace cache. Purpose is power savings (80% of time, decode off).

# Back End




Can retire up to 4 micro-ops/cycle. Only 25% more than Pentium IV. However, Sandy Bridge does so much more often …

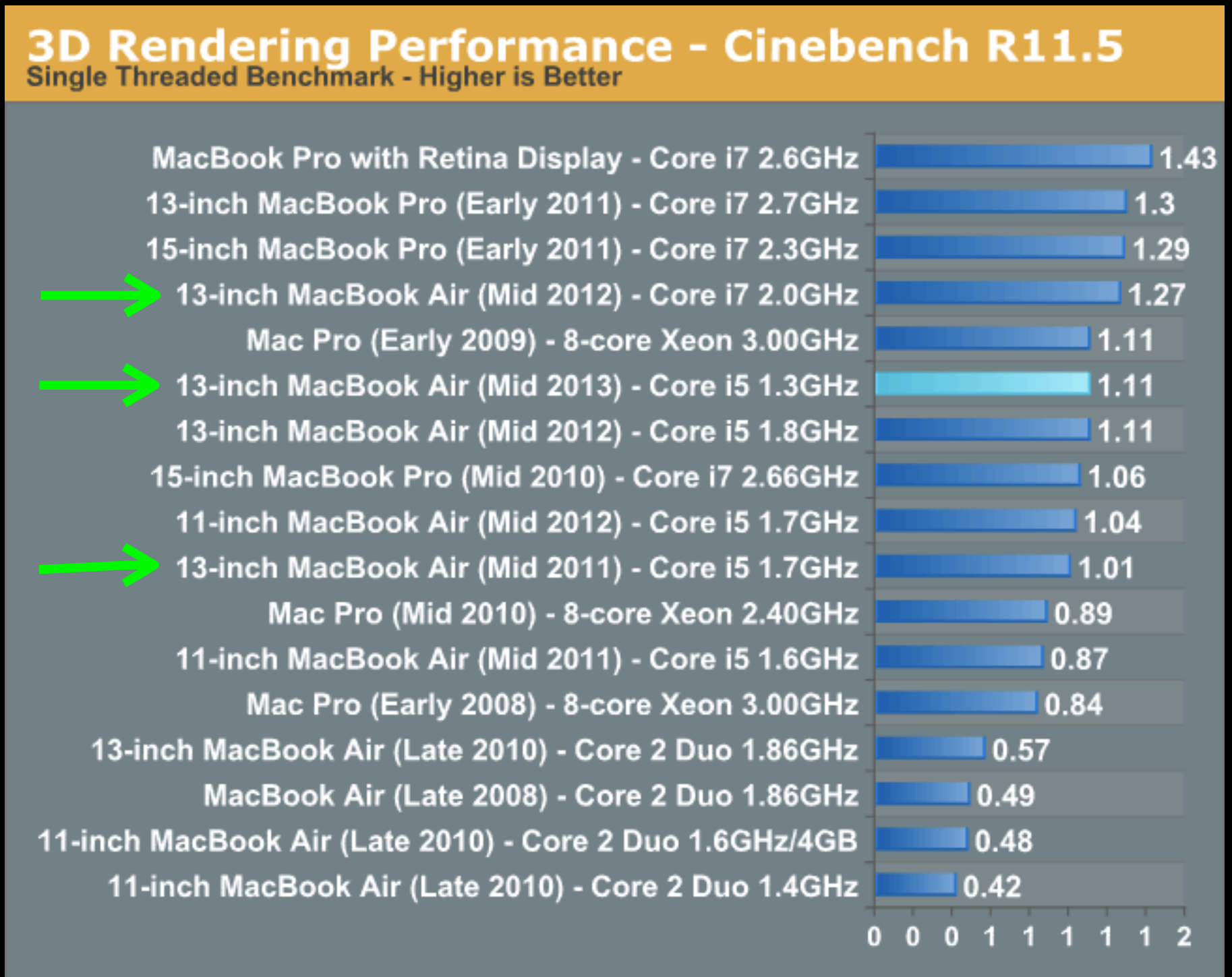
| | Nehalem | Sandy Bridge |
|---|---|---|
| **Load Buffers** | 48 entries | 64 entries |
| **Store Buffers** | 32 entries | 36 entries |
| **Scheduler Entries** | 36 micro-ops | 54 micro-ops |
| **Integer Rename File** | Not applicable | 160 registers |
| **FP Rename File** | Not applicable | 144 registers |
| **Reorder Buffer** | 128 micro-ops | 168 micro-ops |

# Single-thread benchmark: Diminishing returns …

Ivy Bridge

Haswell

Sandy Bridge

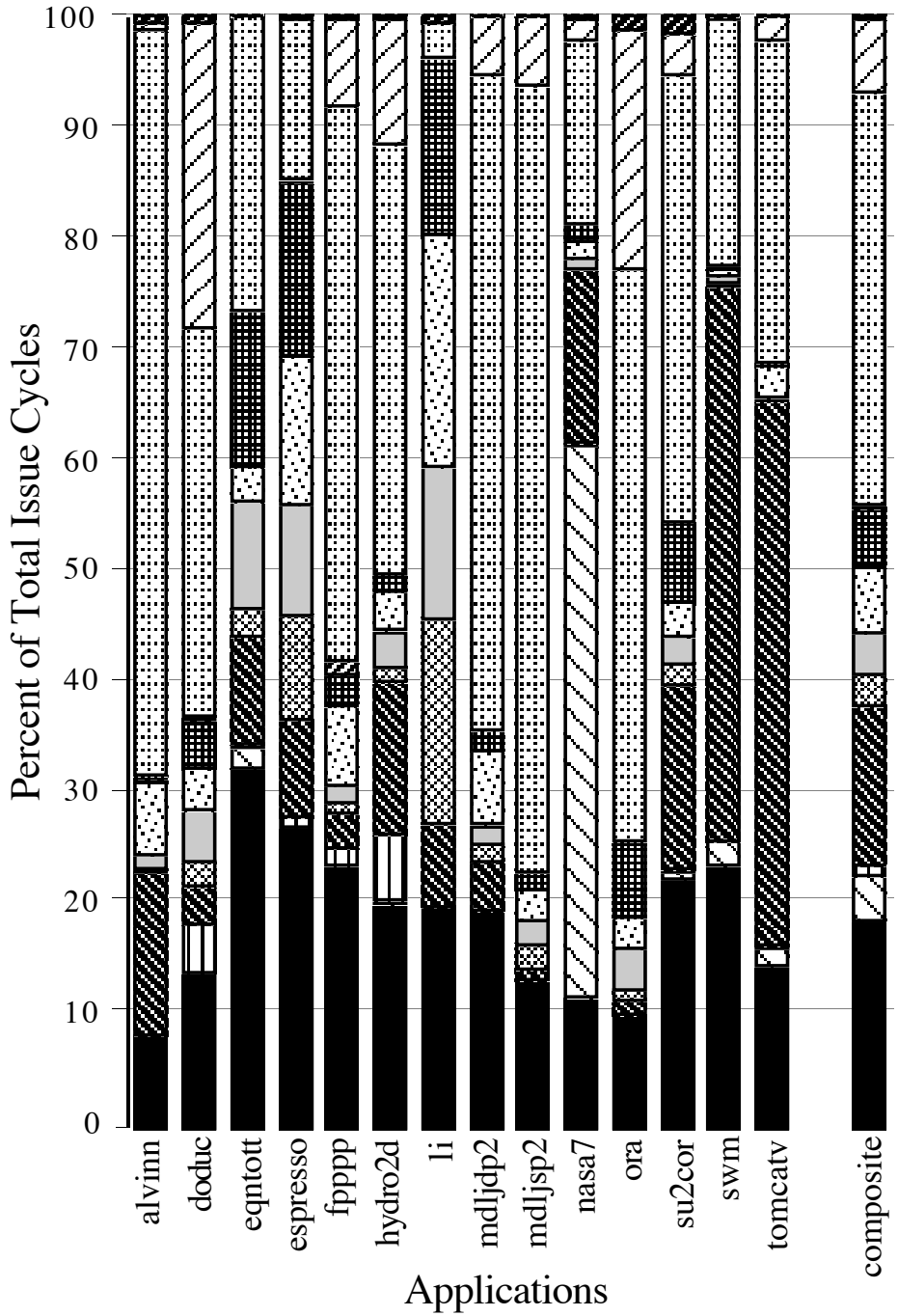


# Limits to Instruction-Level Parallelism

# Recall: Most execution units lie idle

**Big Question**

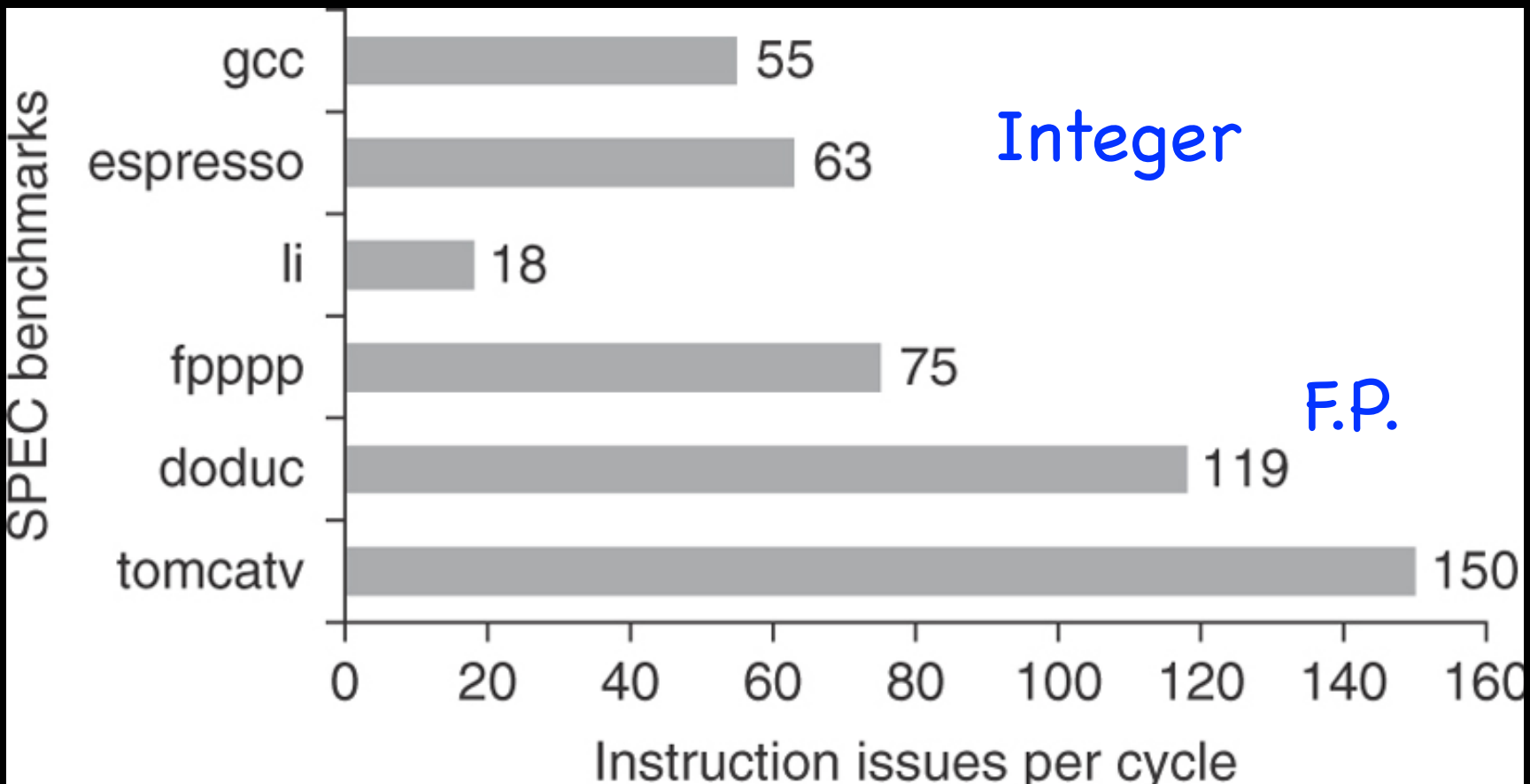
**Is there instruction level parallelism in single threads yet to be attained? Or are we near fundamental limits?**




**For an 8-way superscalar.**

From: Tullsen, Eggers, and Levy, "Simultaneous Multithreading: Maximizing On-chip Parallelism, ISCA 1995.

Instruction level parallelism available for selected SPEC benchmarks, given a perfect architecture.




Perfect? Branch predictor accuracy of 100%, caches never miss, infinite out-of-order resources ...

# Add minor real-world constraints

Reorder buffer windows as shown.

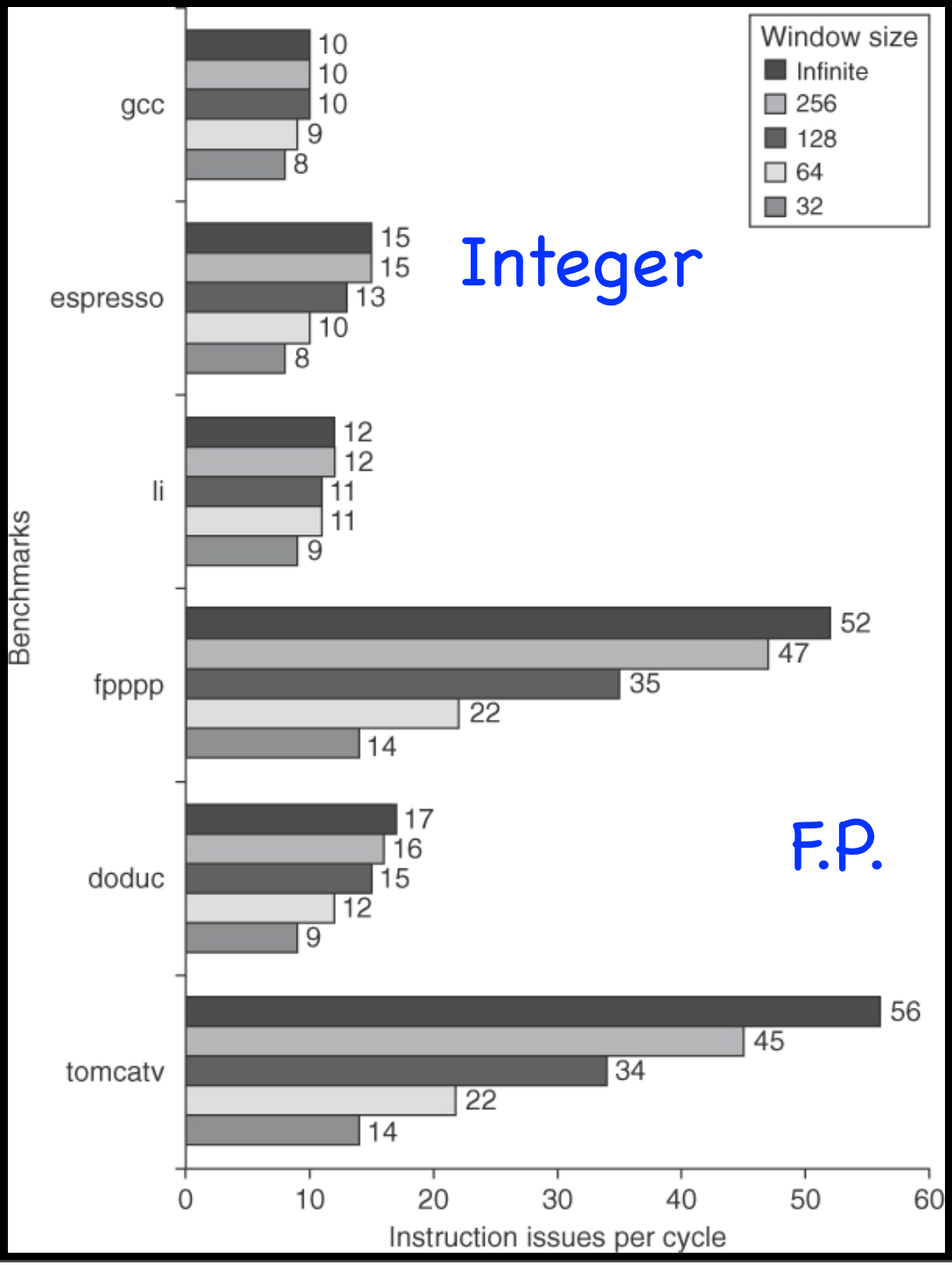
64 issues/cycle max

64 physical registers for int and for float.

Branch predictors near the current state-of-the-art.

Infinite load/store buffers.

Perfect caches

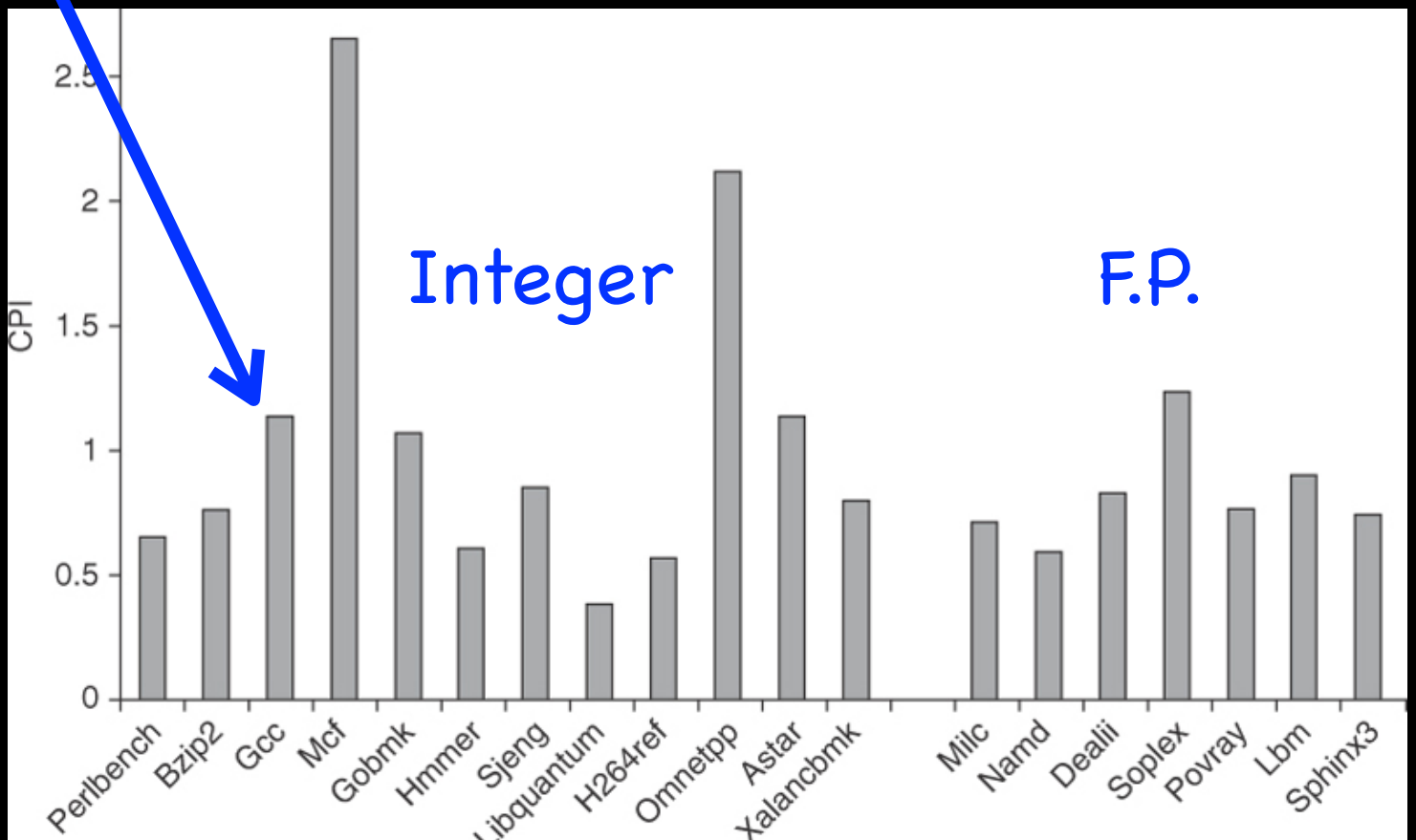


# Actual CPIs on Intel Nehalem

Gcc: Perfect IPC was 55.

"Real-world" machines were 8-10.

Intel Nehalem was slightly under 1.




Maybe new ideas are needed ... maybe incremental improvements will eventually get us there ... or maybe we should focus on other goals than single-threaded performance.

# On Thursday

Beyond Von Neumann ...

| | | |
|---|---|---|
| Th 4/10 | Dataflow | |

Have fun in section !