ECE 454 Computer Systems Programming

CPU Architectures

Ashvin Goel, Ding Yuan
ECE Dept, University of Toronto

Content

- Examine the tricks CPUs play for efficient execution
  - History of CPU architectures
  - Basics of modern CPU architectures

- More details are covered in ECE 552

Before we Start…

- Isn't the CPU speed merely driven by transistor density?
  - Transistor density increase → clock cycle increase → faster CPU

True

True, but
there is more…

- A faster CPU requires
  - Faster clock cycle
  - Smaller cycles per instruction (CPI)
    - ***CPI is the focus of this lecture!***

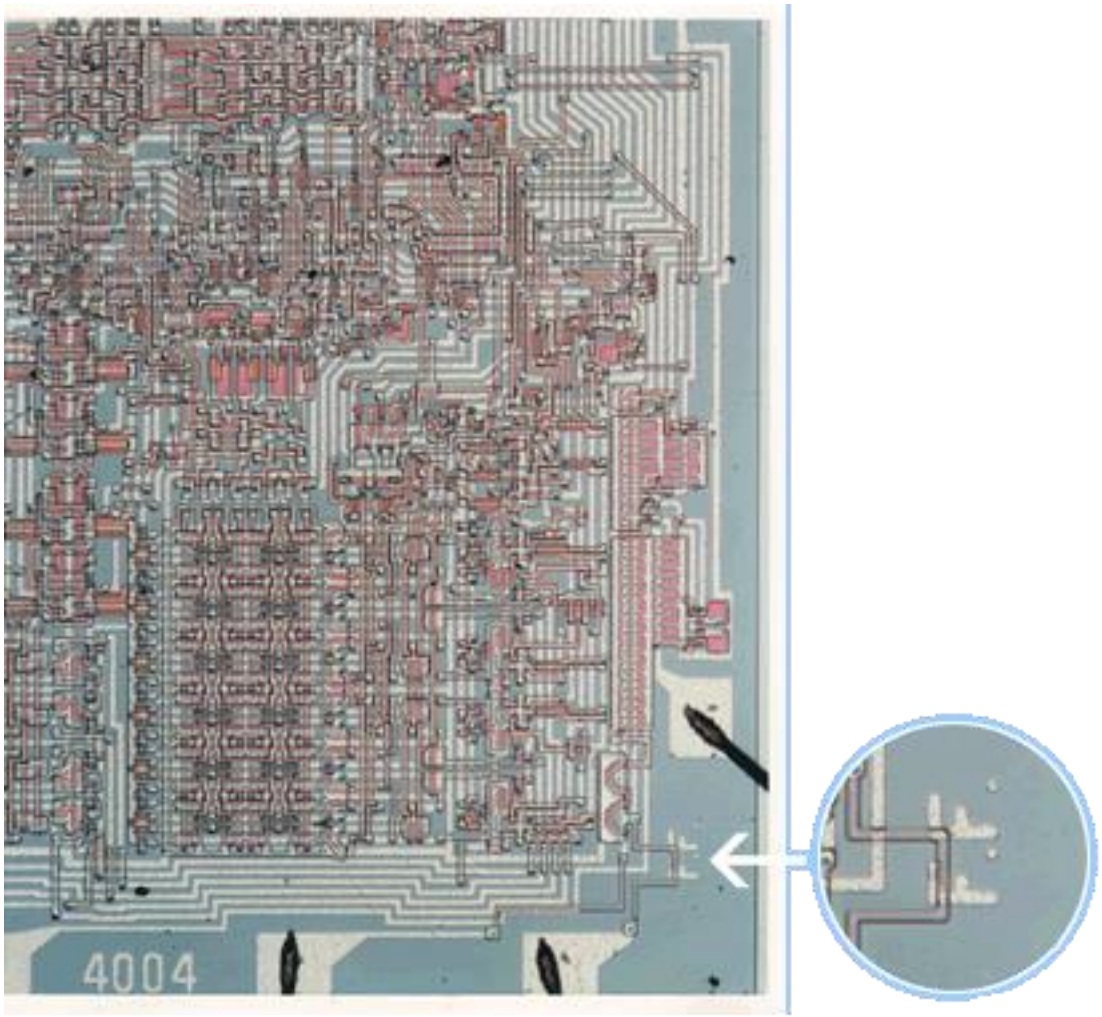
In the Beginning…

- 1961:
  - First commercially-available integrated circuits
  - By Fairchild Semiconductor and Texas Instruments
- 1965:
  - Gordon Moore's observation: (director of Fairchild research)
    - Number of transistors on chips was doubling every two years

1971: Intel Releases the 4004

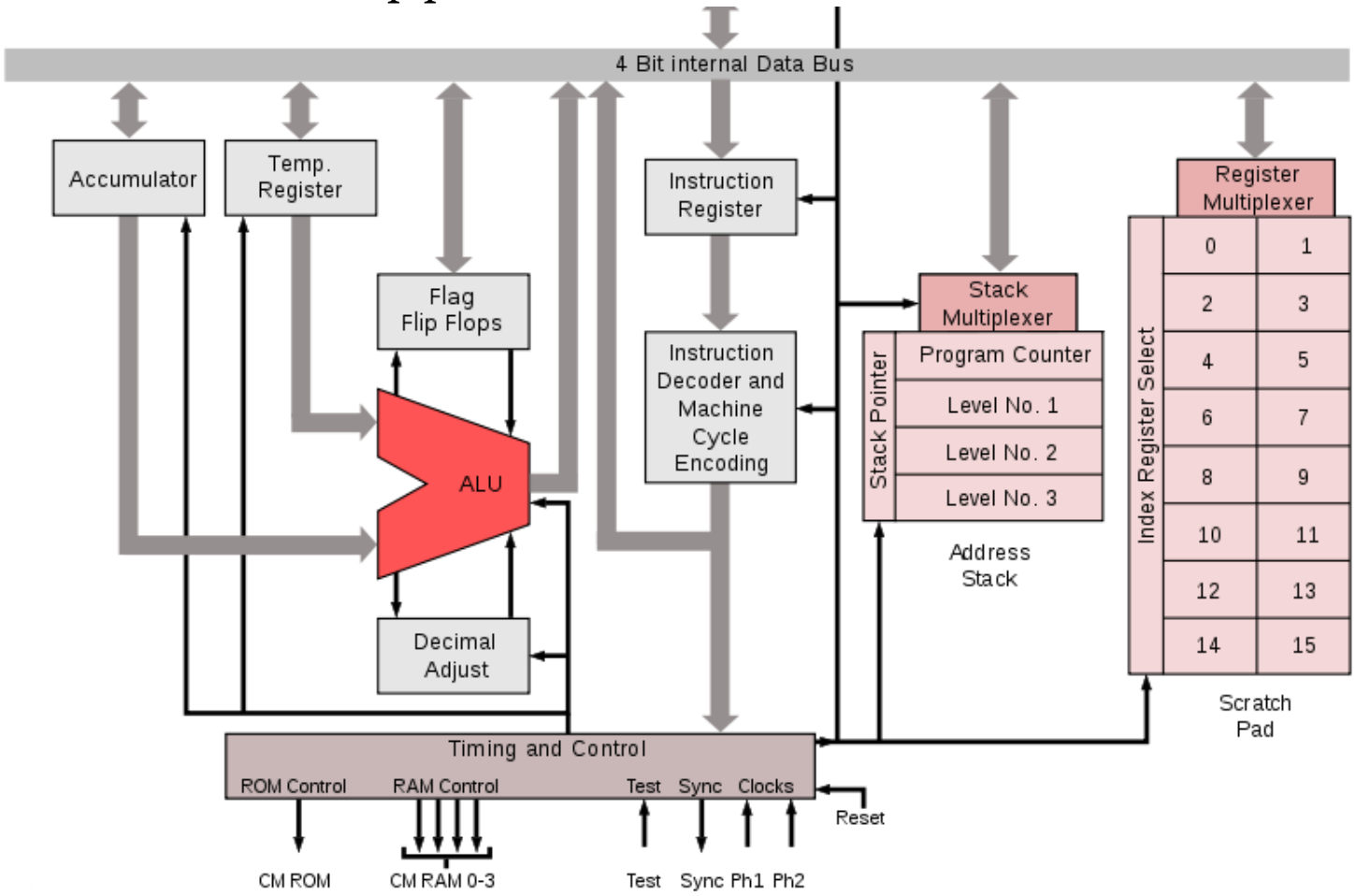
- First commercially available, stand-alone microprocessor
- 4-bit processor; 108KHz; 2300 transistors
- For use in calculators
  - 4 chips: 4004 CPU, 4001 ROM, 4002 RAM, I/O registers

Designed by
Federico Faggin



Intel 4004 (first microprocessor)

- 4-bit processor, but 4KB ROM (how)?
- 3 Stack registers (what does this mean)?
- No Virtual Memory support
- No Interrupt
- No pipeline



The 1970's (Intel): Increased Integration

4004

- 1971: 108KHz; 2300 trans.;
  - 4-bit processor for use in calculators

8008

- 1972: 500KHz; 3500 trans.; 20 support chips
  - 8-bit general-purpose processor

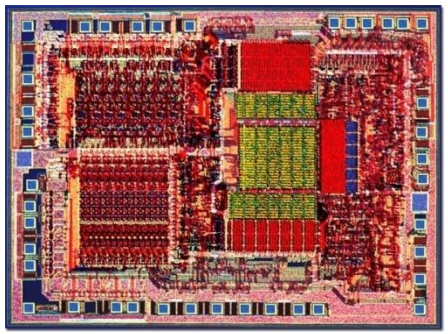
8080

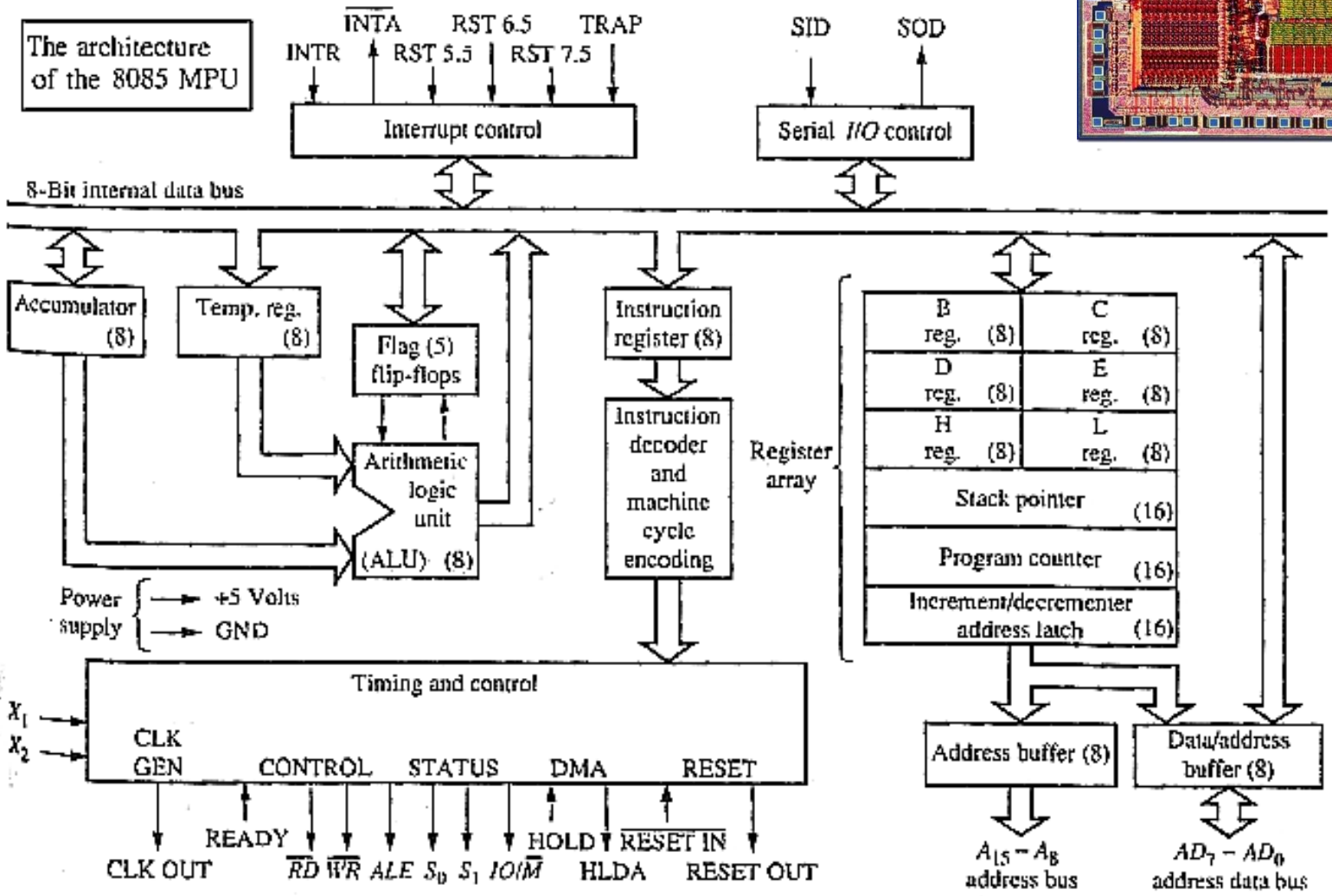
- 1974: 2MHz; 6k trans.; 6 support chips
  - 16-bit addr space, 8-bit registers, used in 'Altair'

8086

- 1978: 10MHz; 29k trans.;
  - Full 16-bit processor, start of x86

Intel 8085

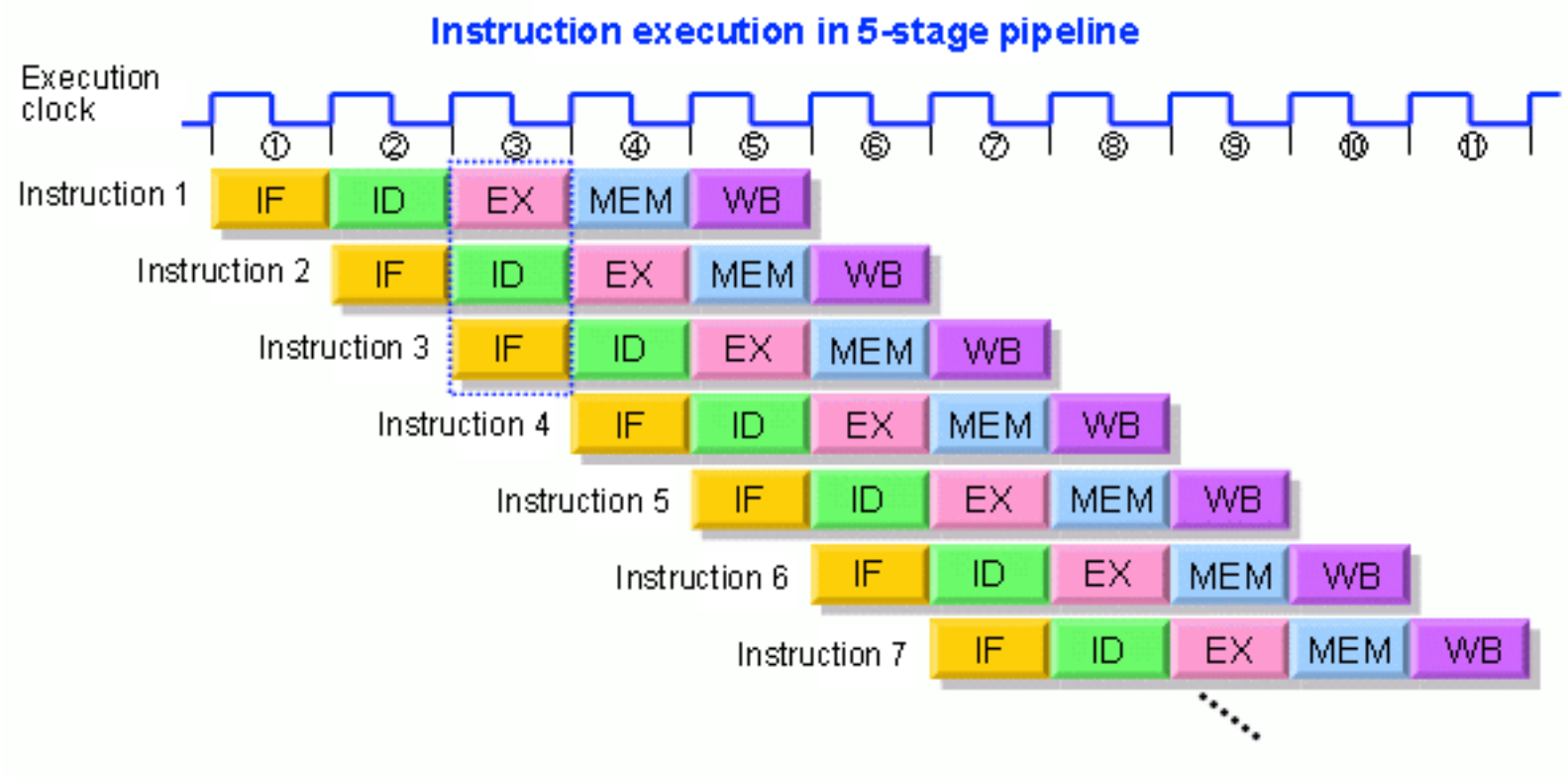



The architecture of the 8085 MPU
INTA
INTR
RST 5.5
RST 6.5
RST 7.5
TRAP
Interrupt control
SID
SOD
Serial I/O control
8-Bit internal data bus
Accumulator (8)
Temp. reg. (8)
Flag (5) flip-flops
Arithmetic logic unit (ALU) (8)
Instruction register (8)
Instruction decoder and machine cycle encoding
Register array
B reg. (8)
C reg. (8)
D reg. (8)
E reg. (8)
H reg. (8)
L reg. (8)
Stack pointer (16)
Program counter (16)
Increment/decrementer address latch (16)
Power supply
+5 Volts
GND
Timing and control
X1
X2
CLK GEN
CONTROL
STATUS
DMA
RESET
READY
HOLD
RESET IN
CLK OUT
RD WR ALE S0 S1 IO/M
HLDA
RESET OUT
Address buffer (8)
Data/address buffer (8)
A15 – A8
address bus
AD7 – AD0
address data bus

The 1980's: RISC and Pipelining

- 1980: Patterson (Berkeley) coins term RISC
  - 1982: Makes RISC-I pipelined processors (only 32 instructions)
- RISC design simplifies implementation
  - Small number of instruction formats
  - Simple instruction processing
- 1981: Hennessy (Stanford) develops MIPS
  - 1984: Forms MIPS computers
- RISC leads naturally to pipelined implementation
  - Partition activities into stages
  - Each stage has simple computation

RISC Pipeline



Reduce CPI from 5 → 1 (ideally)

1985: Pipelining: Intel 386

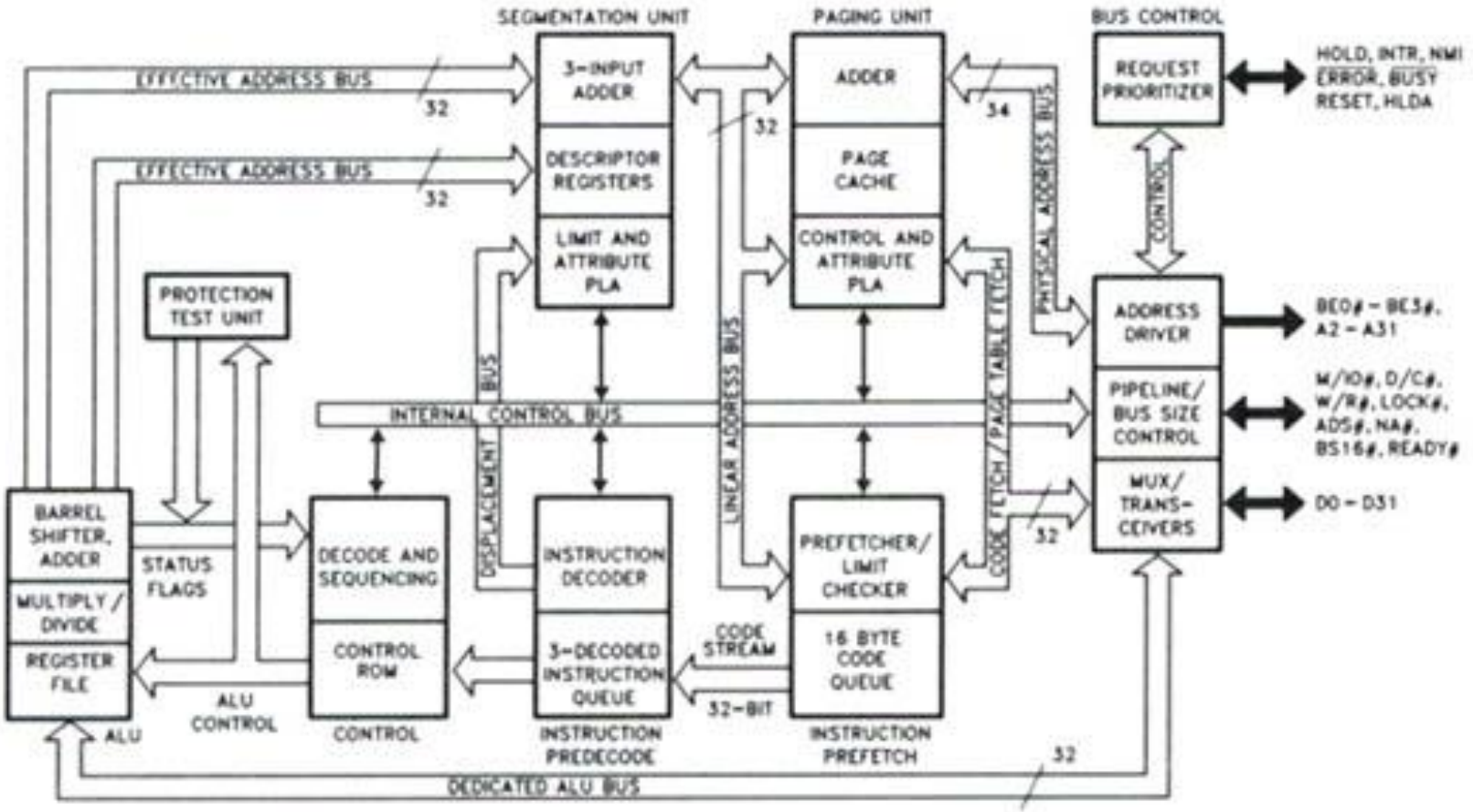
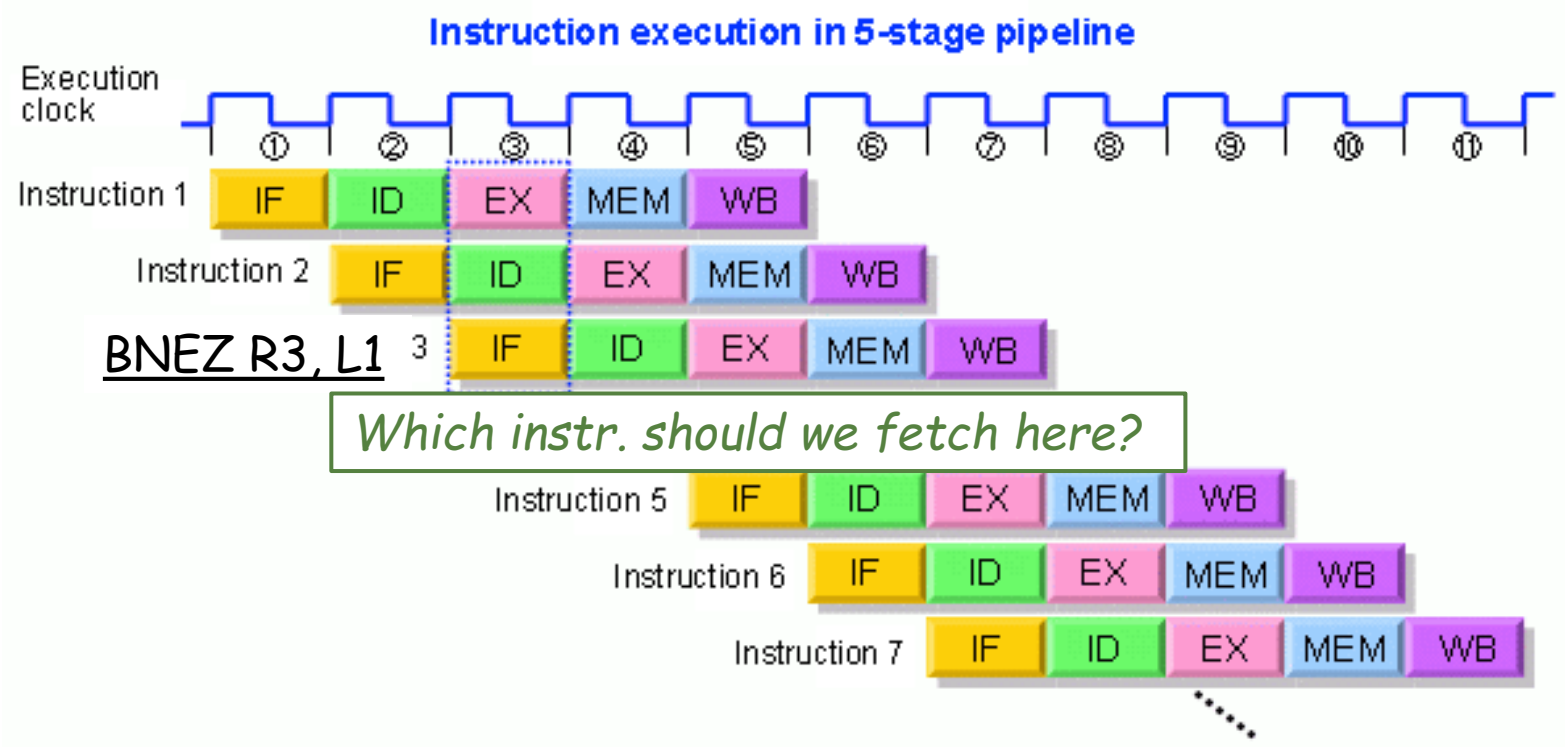


Figure 1-1. 386DX™ Microprocessor Pipelined 32-Bit Microarchitecture

33MHz, 32-bit processor

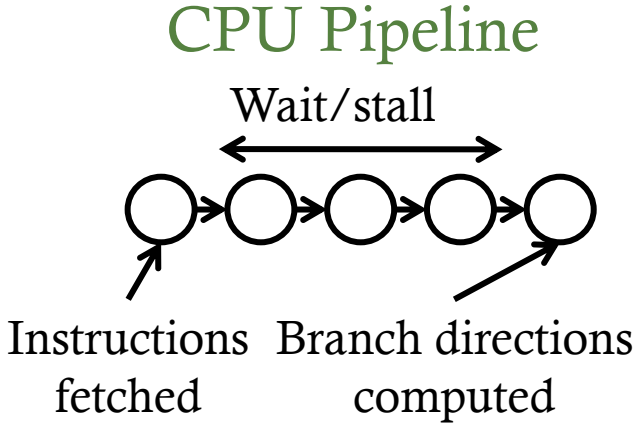
Pipelines and Branch Prediction



- Must wait/stall fetching until branch direction known?
- Solutions?

Pipelines and Branch Prediction

- How bad is the problem? (isn't it just one cycle?)
  - Branch instructions: 15% - 25%
  - Making pipeline deeper
    - Cycles are smaller but branch not resolved until much later
    - => *Misprediction penalty larger*
  - Superscalar architecture
    - Multiple instructions issued simultaneously
    - => *Requires flushing and refetching more instructions*
  - Object-oriented programming
    - => *More indirect branches, making it harder to predict*



Branch Prediction: Solution

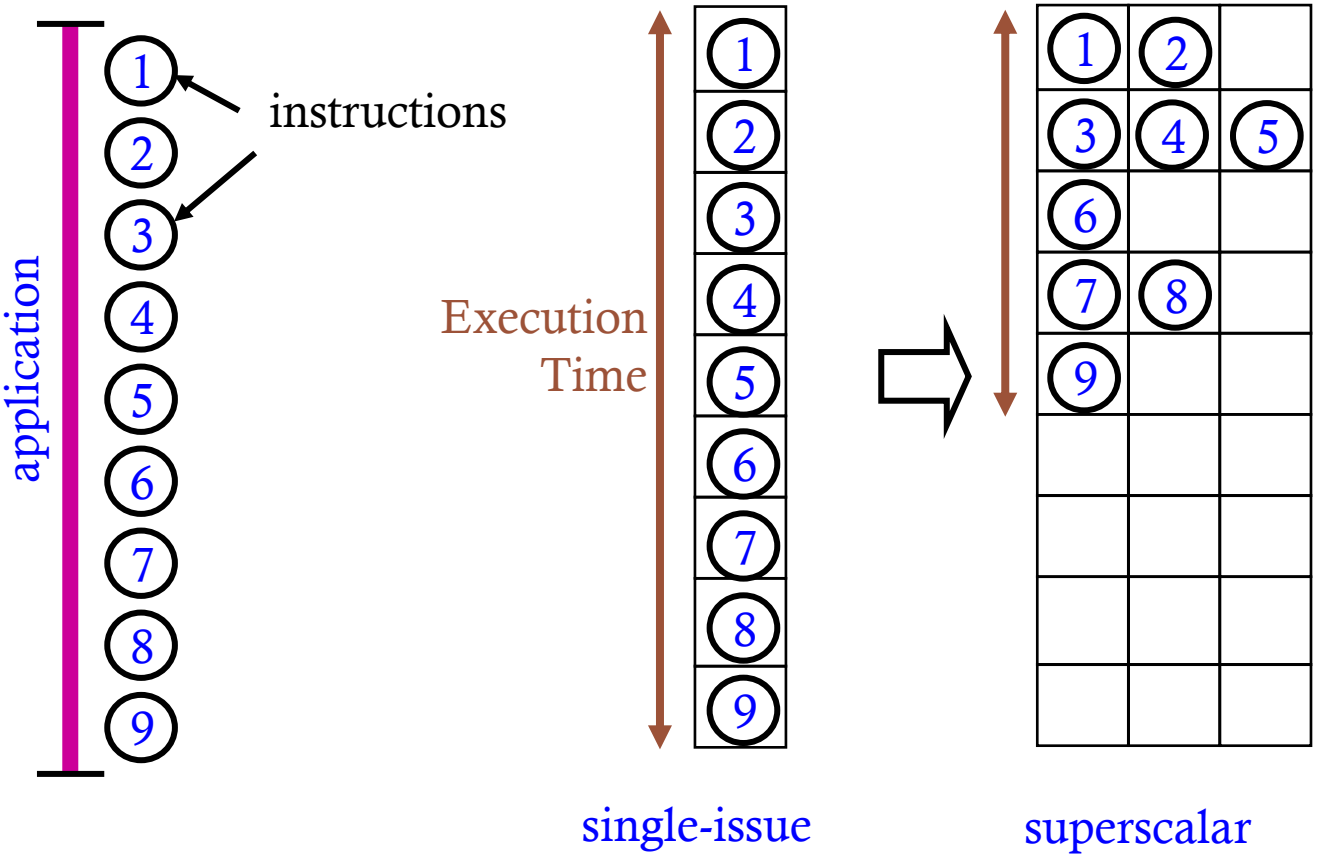
- Solution: predict branch directions:
  - Intuition: predict the *future* based on *history*
  - Use a table to remember outcomes of previous branches

BP is important: prediction tables uses about 30K bits on Intel P4!

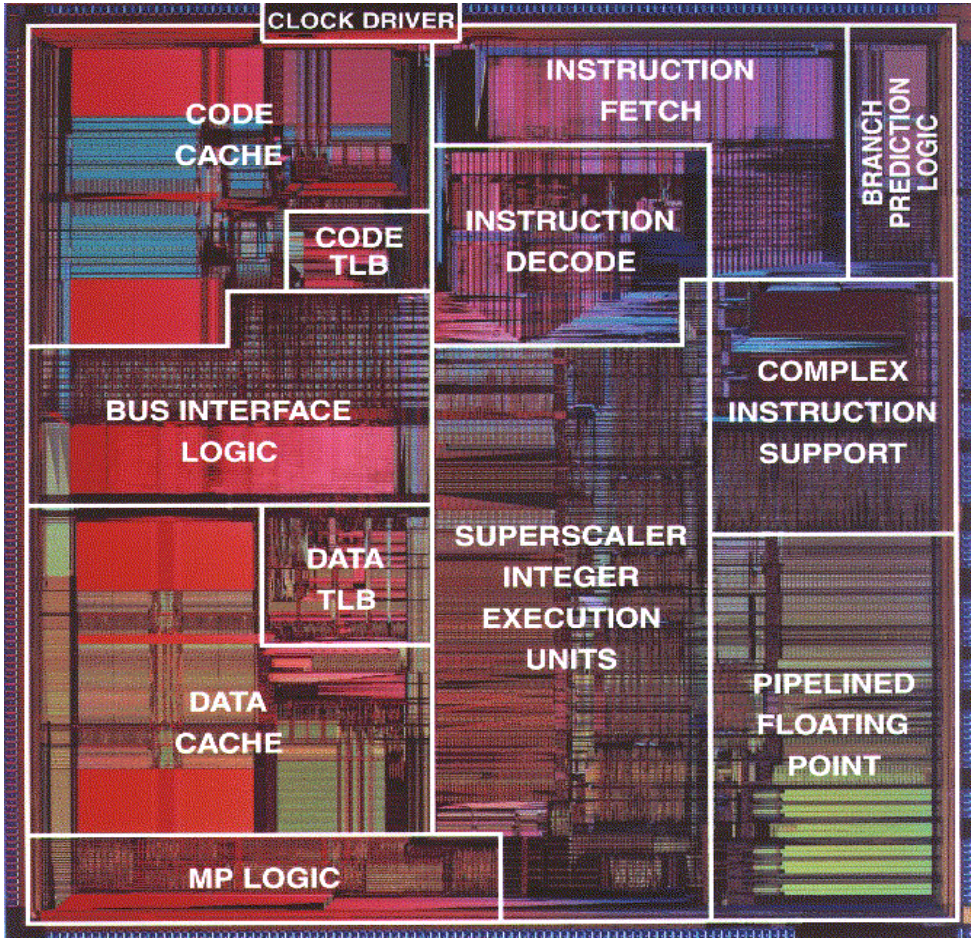
What Do We Have So Far?

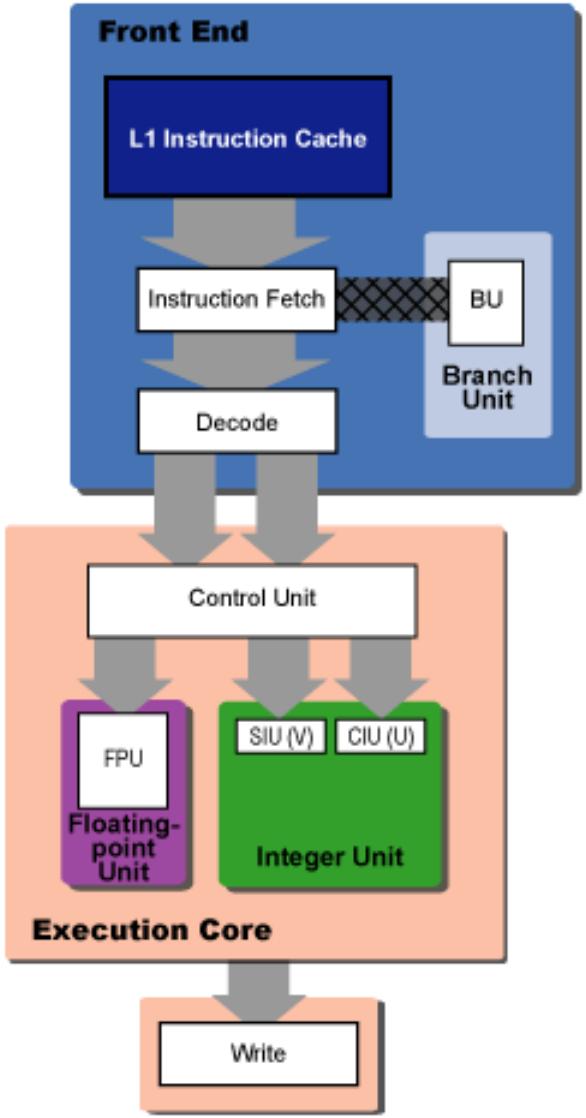
- CPI:
  - Pipeline: reduce CPI from *n* to *1* (ideal case)
  - Branch instruction will cause stalls: *effective CPI > 1*
    - Branch prediction
- *But can we reduce CPI to <1?*

Instruction-Level Parallelism



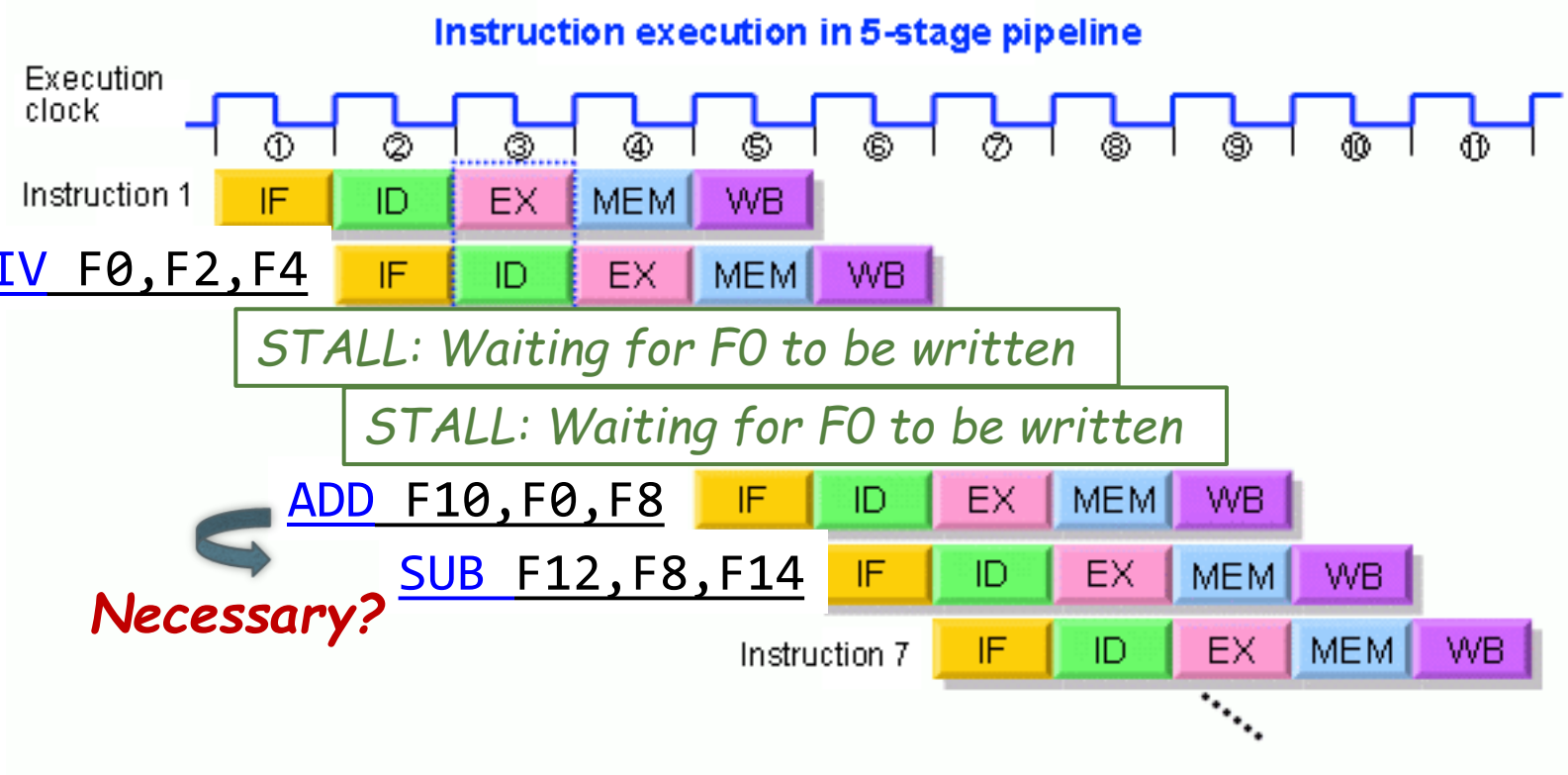
1993: Intel Pentium





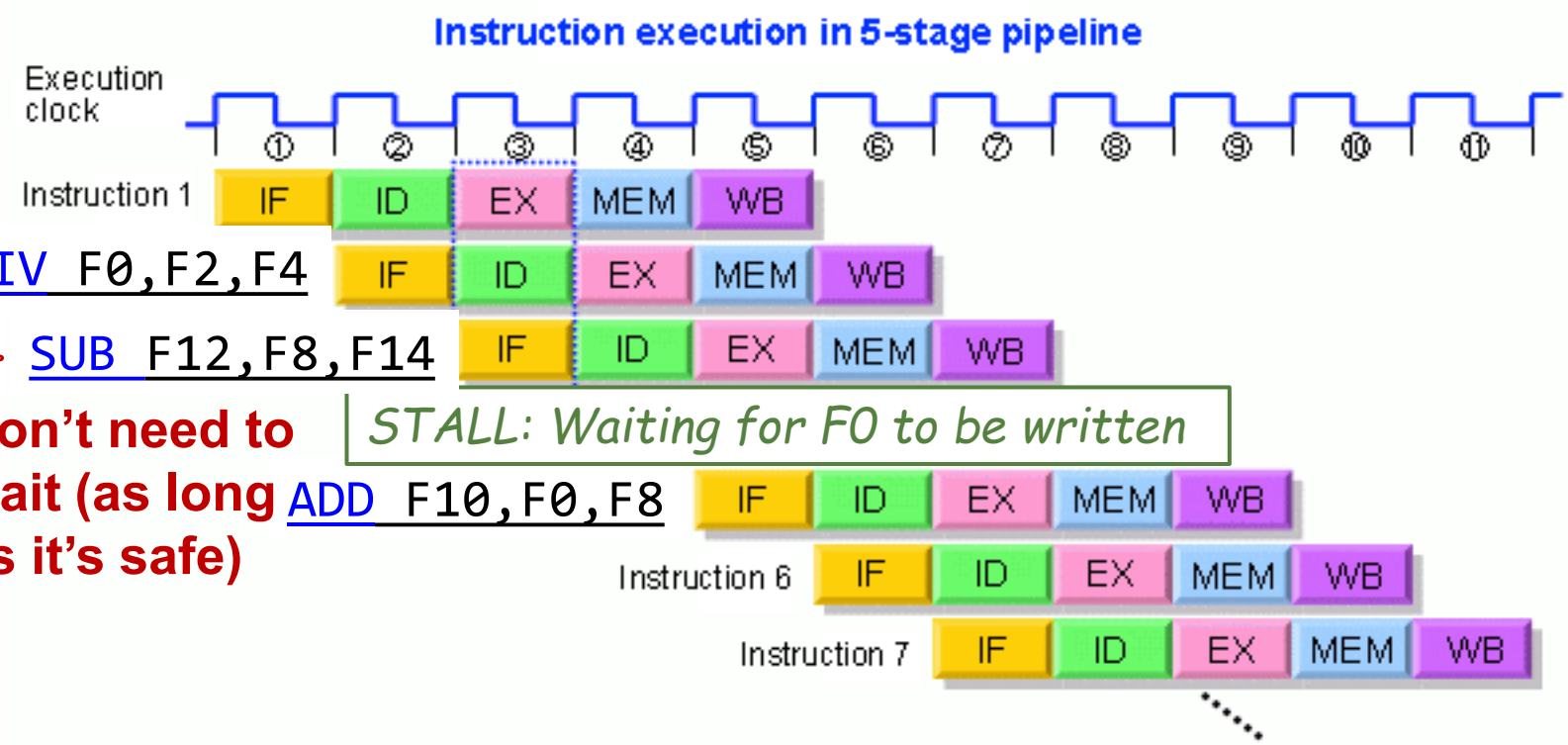
Data Hazard: Obstacle to Perfect Pipelining

```
DIV  F0, F2, F4 // F0 = F2/F4
ADD  F10, F0, F8 // F10 = F0 + F8
SUB  F12, F8, F14 // F12 = F8 - F14
```



Out-of-order Execution: Solving Data-Hazard

```
DIV  F0, F2, F4 // F0 = F2/F4
ADD  F10, F0, F8 // F10 = F0 + F8
SUB  F12, F8, F14 // F12 = F8 – F14
```



Don't need to wait (as long as it's safe)

Out-of-Order Execution to Mask Cache Miss Delay

IN-ORDER:

inst1
inst2
inst3
inst4
load (misses cache)

Cache miss latency

inst5 (must wait for load value)
inst6

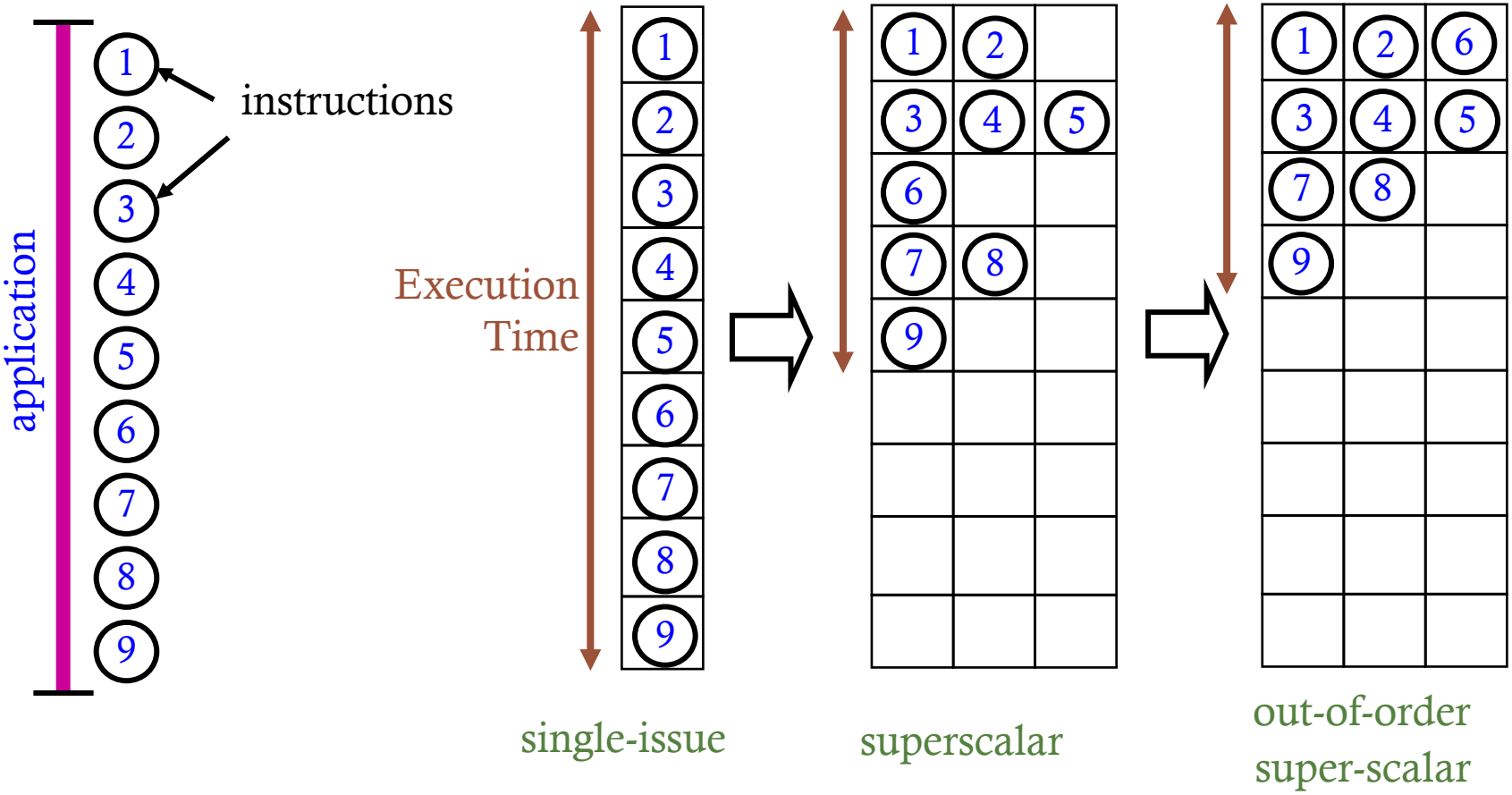
OUT-OF-ORDER:

inst1
load (misses cache)
inst2
inst3
inst4

Cache miss latency

inst5 (must wait for load value)
inst6

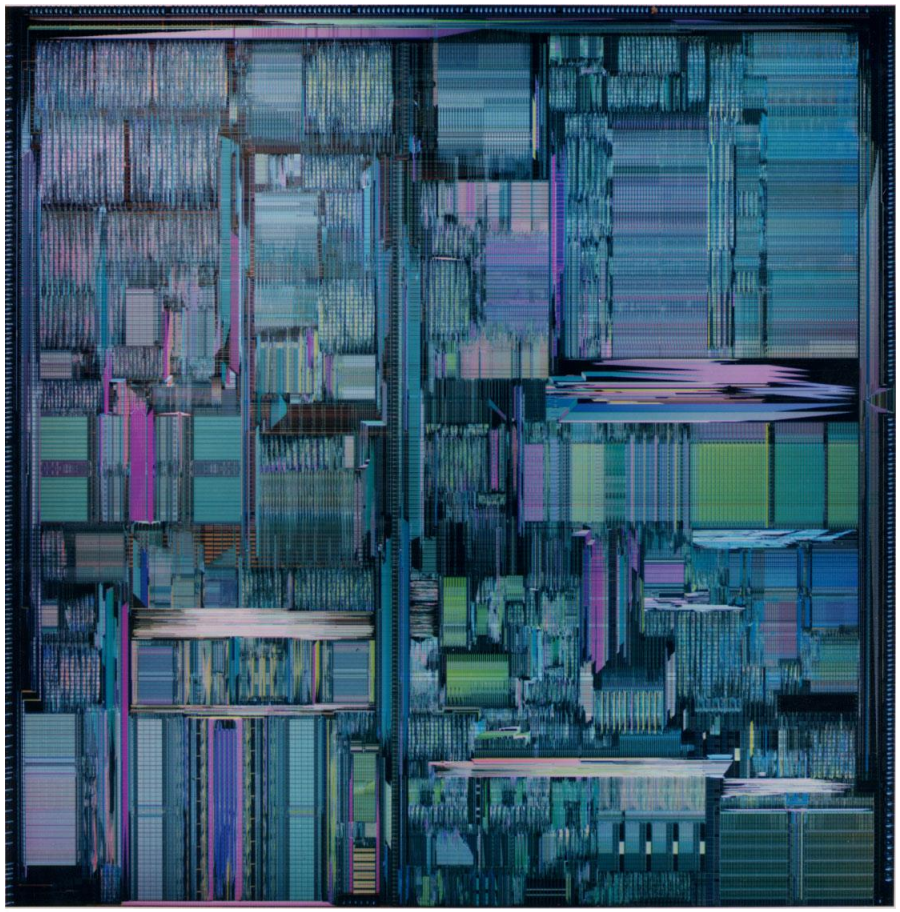
Instruction-level Parallelism + Out-of-Order Execution

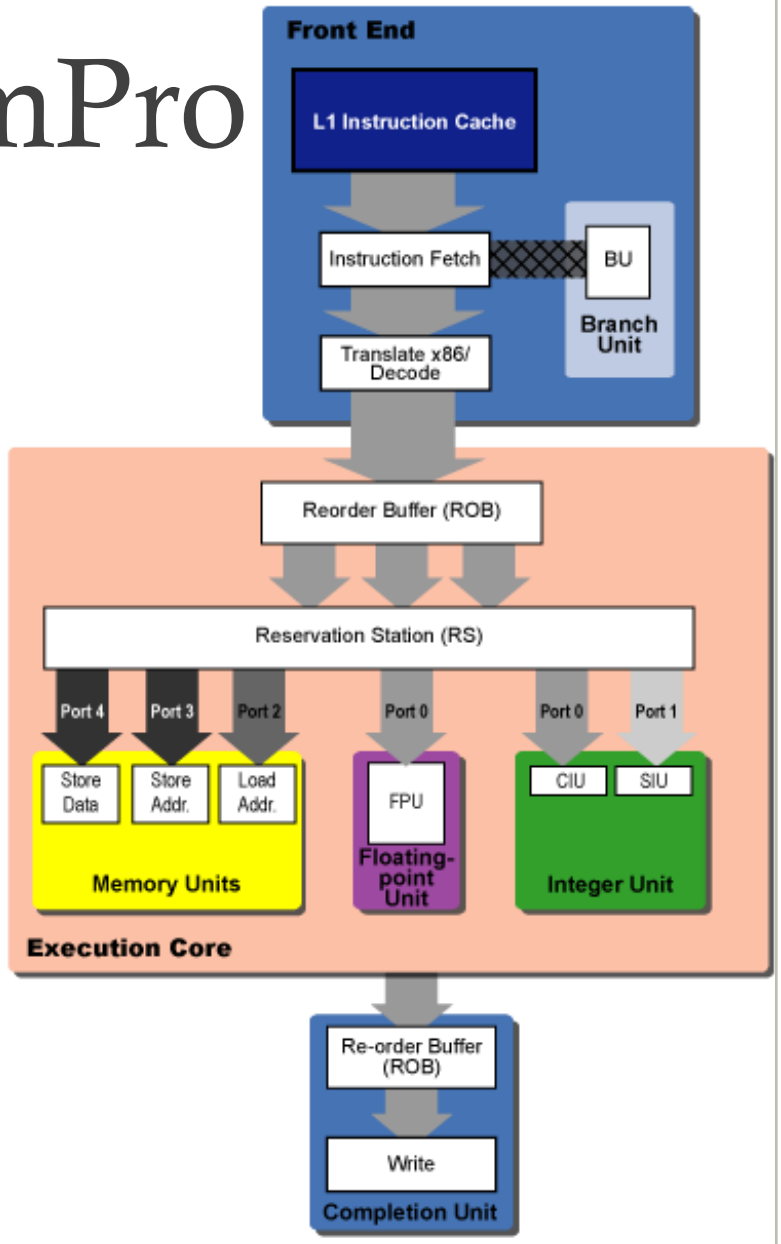


Out-of-Order Execution

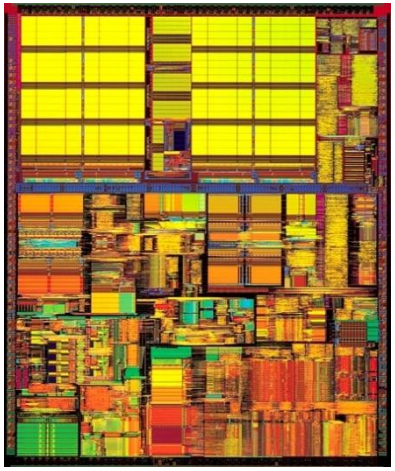
- In practice, much more complicated since we need to detect and stall for all dependencies or else program will execute incorrectly
  - E.g., what if I write to a register too early?
  - E.g., what if interrupts and exceptions occurs in between

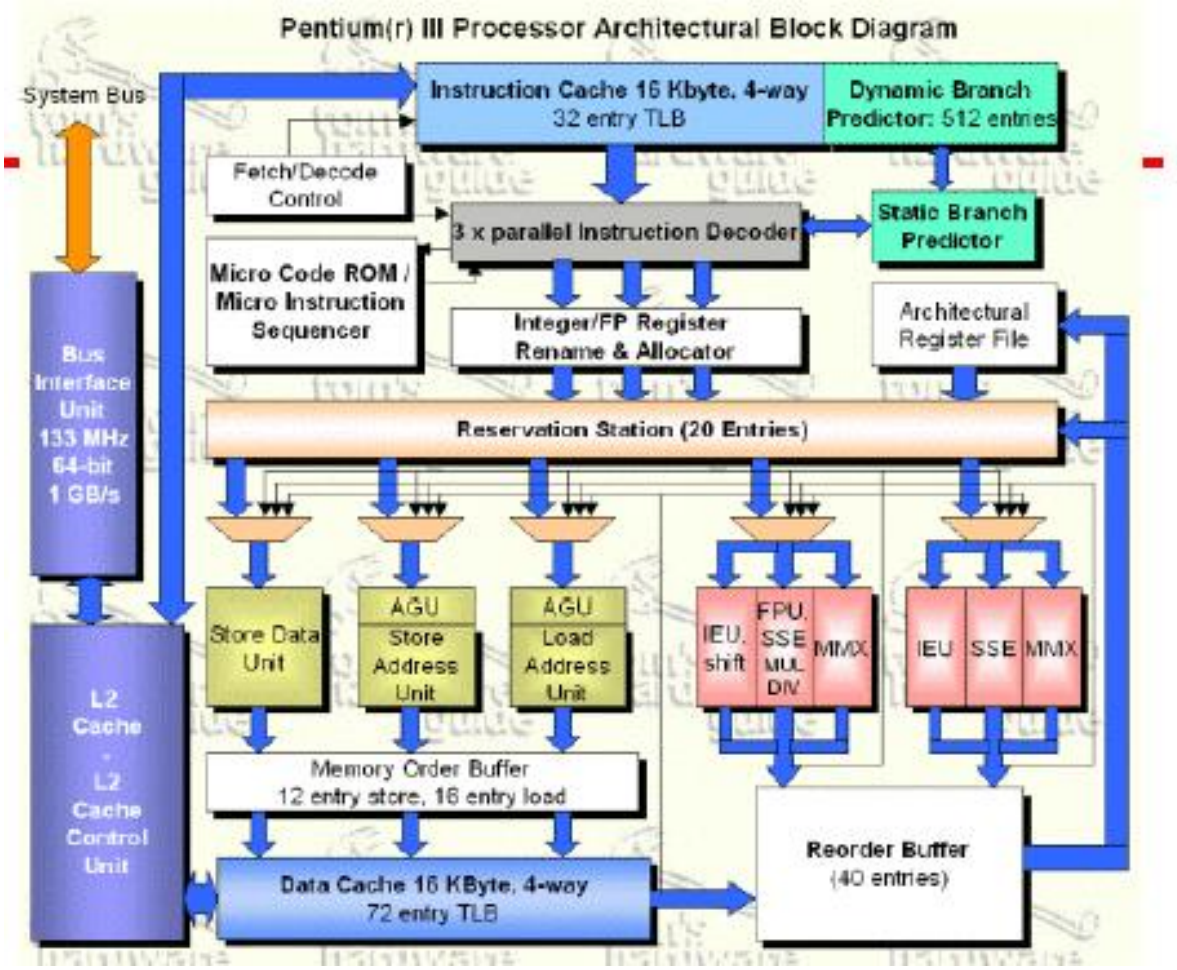
1995: Intel PentiumPro



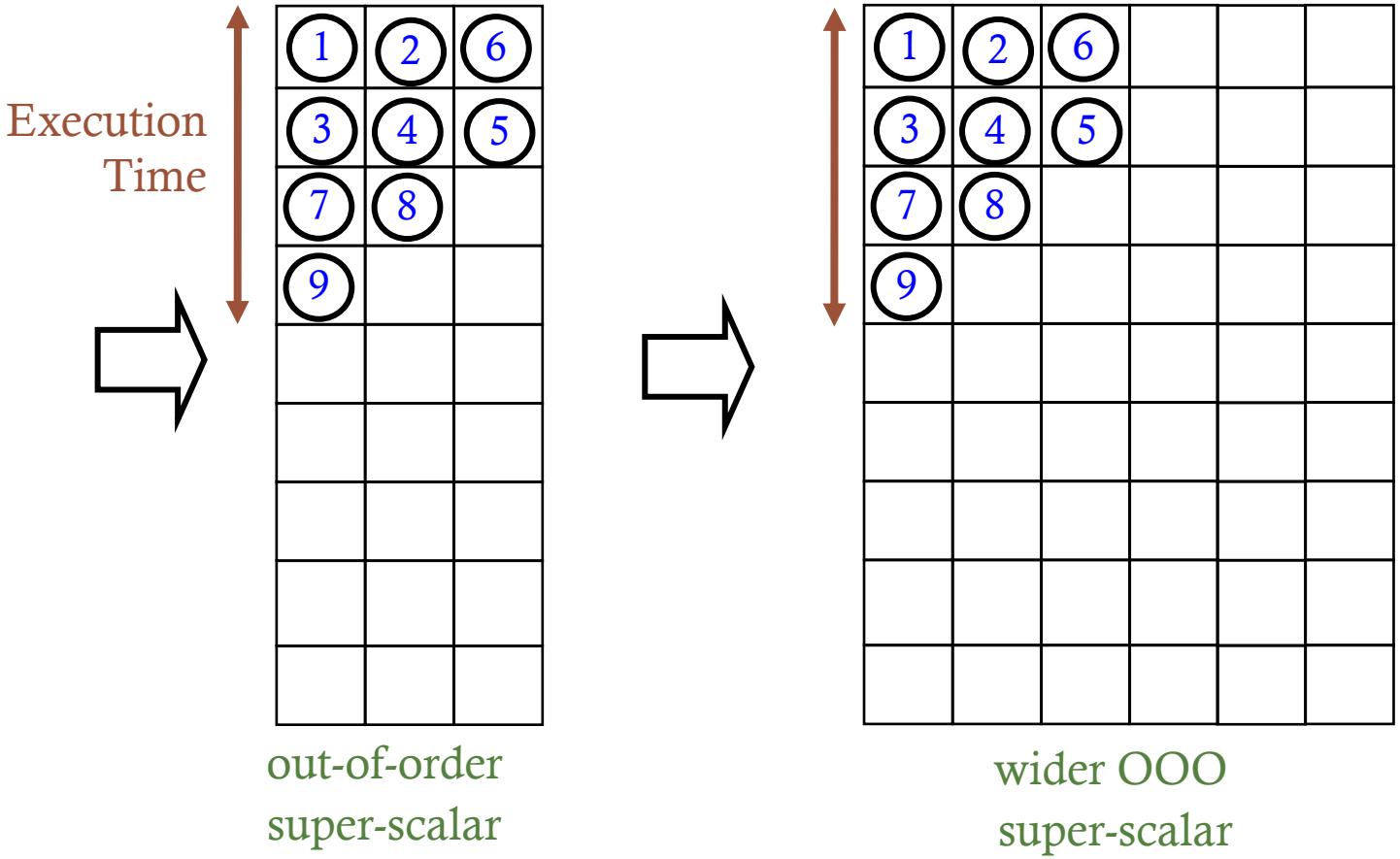


1999: Pentium III



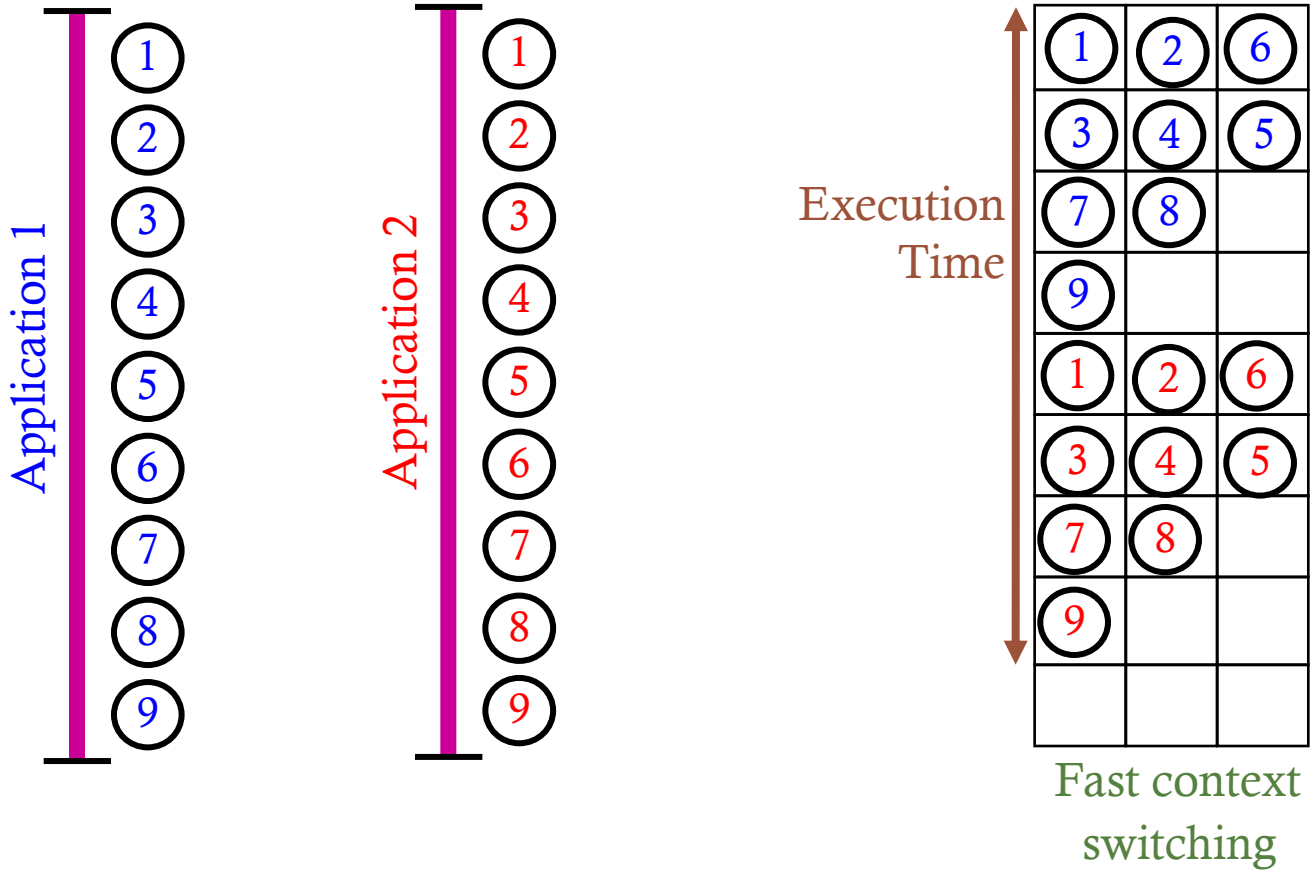


The Limits of Instruction-Level Parallelism

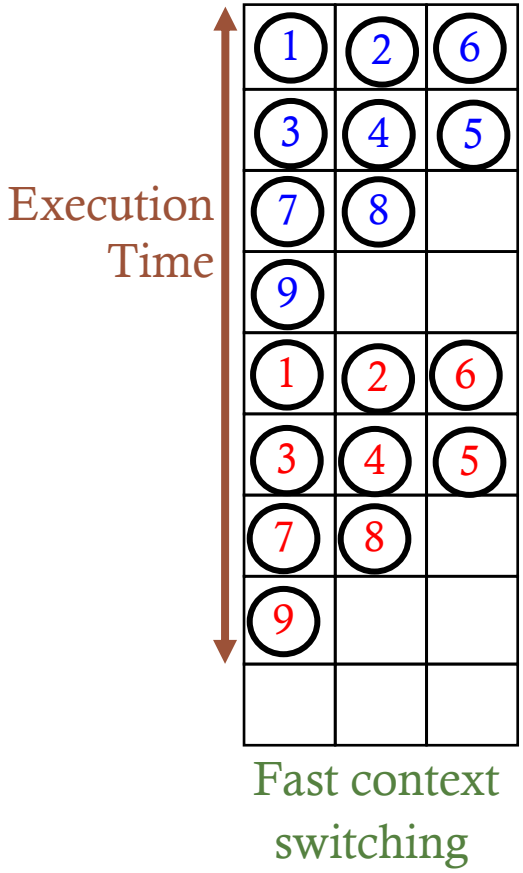


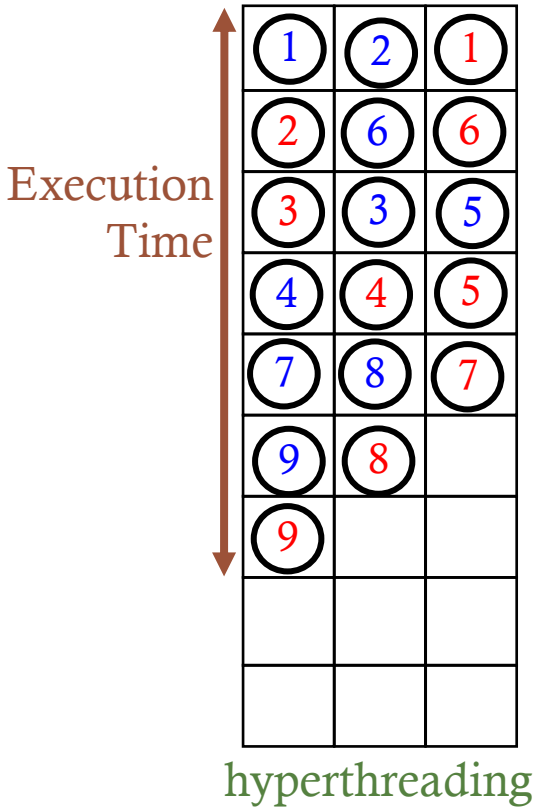
☞**Diminishing returns for wider superscalar**

Multithreading The "Old Fashioned" Way



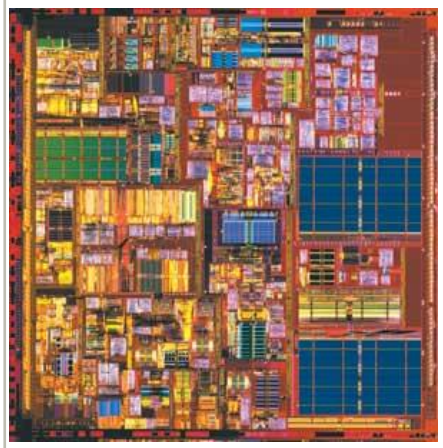
Simultaneous Multithreading (SMT) (aka Hyperthreading)

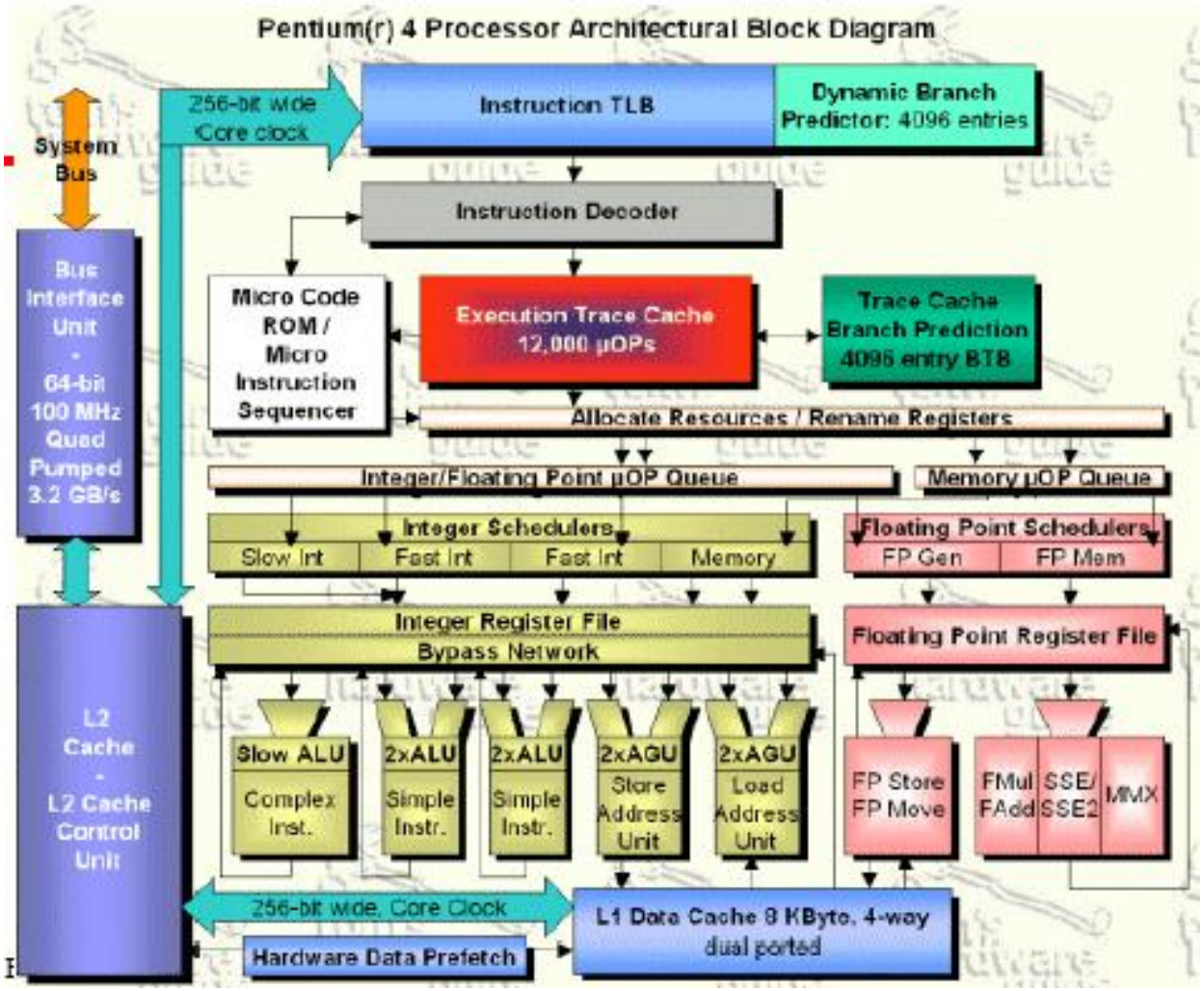




☞ **SMT: 20-30% faster than context switching**

2000: Pentium IV





Putting it All Together: Intel

Year	Processor	Tech.	CPI
1971	4004	No pipeline	*n*
1985	386	Pipelining Branch prediction	*close to 1* *closer to 1*
1993	Pentium	Superscalar	*< 1*
1995	PentiumPro	Out-of-order exec.	*<< 1*
1999	Pentium III	Deep pipeline	*shorter cycle*
2000	Pentium IV	SMT	*<<<1*

32-bit to 64-bit Computing

- Why 64 bit?
  - 32b addr space: 4GB; 64b addr space: 4GB * 4GB = 16M TB
    - Benefits large databases and media processing
  - OSs and counters
    - 64bit counter will not overflow (if doing ++)
  - Math and cryptography
    - Better performance for large/precise value math
- Drawbacks:
  - Pointers now take 64 bits instead of 32
    - I.e., code size increases
  - Unlikely to go to 128 bits