Lecture 3
Instruction Level Parallelism (1)

EEC 171 Parallel Architectures

John Owens

UC Davis

Credits

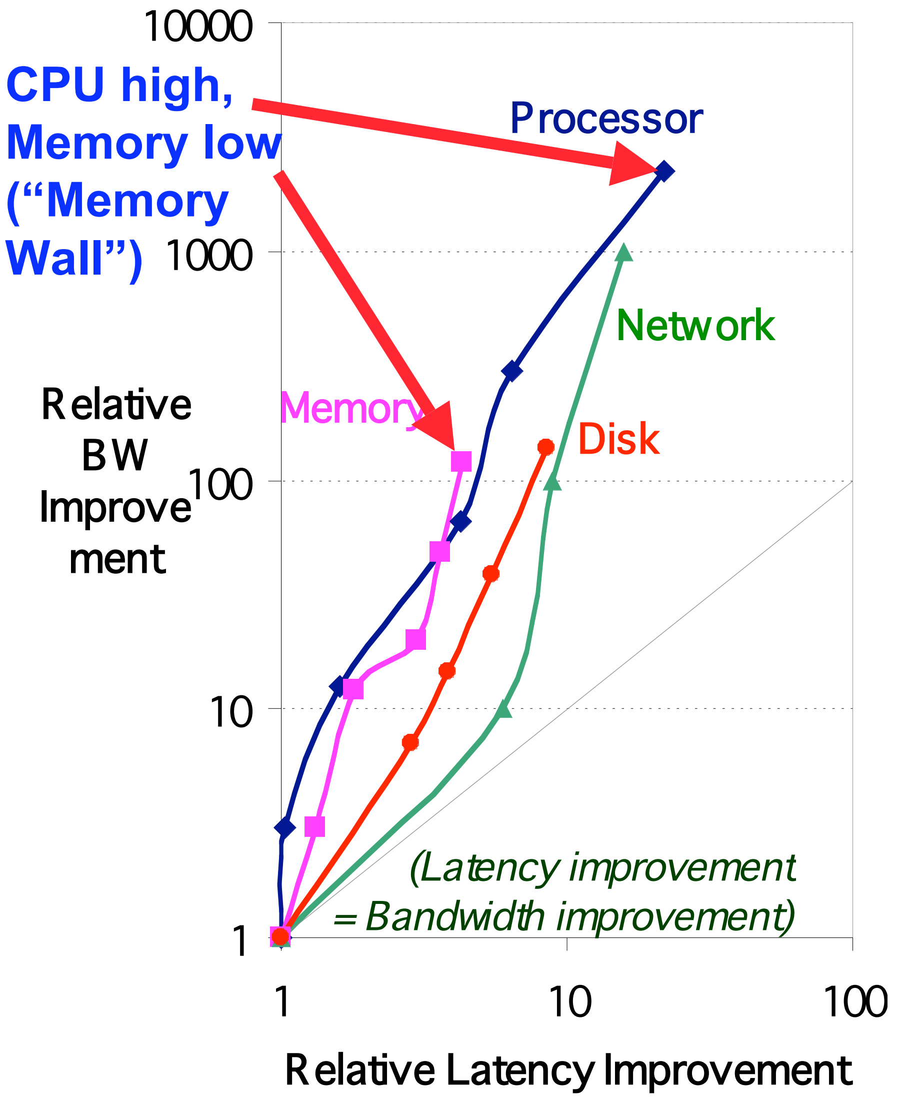
- © John Owens / UC Davis 2007–17.
- Thanks to many sources for slide material: Computer Organization and Design (Patterson & Hennessy) © 2005, Computer Architecture (Hennessy & Patterson) © 2007, Inside the Machine (Jon Stokes) © 2007, © Dan Connors / University of Colorado 2007, © Wen-Mei Hwu/David Kirk, University of Illinois 2007, © David Patterson / UCB 2003–6, © Mary Jane Irwin / Penn State 2005, © John Kubiatowicz / UCB 2002, © Krste Asinovic/Arvind / MIT 2002, © Morgan Kaufmann Publishers 1998.

Today’s Goals

- What is instruction-level parallelism?
- What do processors do to extract ILP?
    - Not “how do they do that” (future lecture)

Finish off lecture from last time

Latency Lags Bandwidth (last ~20 years)



- Processor: '286, '386, '486, Pentium, Pentium Pro, Pentium 4 (21x, 2250x)

Rule of Thumb for Latency Lagging BW

- In the time that bandwidth doubles, latency improves by no more than a factor of 1.2 to 1.4
  - (and capacity improves faster than bandwidth)
- Stated alternatively:

  Bandwidth improves by more than the square of the improvement in latency

6 Reasons Latency Lags Bandwidth

1. Moore's Law helps BW more than latency
    - Faster transistors, more transistors, more pins help bandwidth
        - MPU Transistors: 0.130 vs. 42 M xtors (300X)
        - DRAM Transistors: 0.064 vs. 256 M xtors (4000X)
        - MPU Pins: 68 vs. 423 pins (6X)
        - DRAM Pins: 16 vs. 66 pins (4X)

6 Reasons Latency Lags Bandwidth

- Moore's Law helps BW more than latency
    - Smaller, faster transistors but communicate over (relatively) longer lines: limits latency improvements
        - Feature size: 1.5 to 3 vs. 0.18 micron (8X,17X)
        - MPU Die Size: 35 vs. 204 mm^2 (ratio sqrt ⇒ 2X)
        - DRAM Die Size: 47 vs. 217 mm^2 (ratio sqrt ⇒ 2X)

6 Reasons Latency Lags Bandwidth (cont'd)

2. Distance limits latency
   - Size of DRAM block ⇒ long bit and word lines ⇒ most of DRAM access time
   - Speed of light and computers on network
   - 1. & 2. explains linear latency vs. square BW?

6 Reasons Latency Lags Bandwidth (cont'd)

3. Bandwidth easier to sell ("bigger = better")

- E.g., 10 Gbits/s Ethernet ("10 Gig") vs. 10 µsec latency Ethernet
- 4400 MB/s DIMM ("PC4400") vs. 50 ns latency
- Even if just marketing, customers now trained
- Since bandwidth sells, more resources thrown at bandwidth, which further tips the balance

6 Reasons Latency Lags Bandwidth (cont'd)

4. Latency helps BW, but not vice versa

- Spinning disk faster improves both bandwidth and rotational latency
    - 3600 RPM ⇒ 15000 RPM = 4.2X
    - Average rotational latency: 8.3 ms ⇒ 2.0 ms
    - Things being equal, also helps BW by 4.2X
- Lower DRAM latency ⇒ More access/second (higher bandwidth)
- Higher linear density helps disk BW (and capacity), but not disk latency
    - 9,550 BPI ⇒ 533,000 BPI ⇒ 60X in BW

6 Reasons Latency Lags Bandwidth (cont'd)

5. Bandwidth hurts latency
    - Queues help bandwidth, hurt latency (Queuing Theory)
    - Adding chips to widen a memory module increases bandwidth but higher fan-out on address lines may increase latency
6. Operating System overhead hurts latency more than Bandwidth
    - Long messages amortize overhead; overhead bigger part of short messages

Why Do Processors Get Faster?

- 3 reasons:
  - More parallelism (or more work per pipeline stage): fewer clocks/instruction [more instructions/cycle]
    - Get WIDER
  - Deeper pipelines: fewer gates/clock
    - Get DEEPER
  - Transistors get faster (Moore's Law): fewer ps/gate
    - Get FASTER

Extracting Yet More Performance

- Two options:
  - Increase the depth of the pipeline to increase the clock rate — superpipelining
    - How does this help performance? (What does it impact in the performance equation?)
  - Fetch (and execute) more than one instruction at one time (expand every pipeline stage to accommodate multiple instructions) — multiple-issue
    - How does this help performance? (What does it impact in the performance equation?)
    - Today's topic! $\frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$

Extracting Yet More Performance

- Launching multiple instructions per stage allows the instruction execution rate, CPI, to be less than 1
  - So instead we use IPC: instructions per clock cycle
    - e.g., a 3 GHz, four-way multiple-issue processor can execute at a peak rate of 12 billion instructions per second with a best case CPI of 0.25 or a best case IPC of 4
  - If the datapath has a five stage pipeline, how many instructions are active in the pipeline at any given time?
  - How might this lead to difficulties?

Superpipelined Processors

- Increase the depth of the pipeline leading to shorter clock cycles (and more instructions “in flight” at one time)
    - The higher the degree of superpipelining, the more forwarding/hazard hardware needed, the more pipeline latch overhead (i.e., the pipeline latch accounts for a larger and larger percentage of the clock cycle time), and the bigger the clock skew issues (i.e., because of faster and faster clocks)
    - We know there are limits to this (6–8 FO4 delays)
        - Recall Steve Keckler graph from benchmark/technology lecture

Superpipelined vs. Superscalar

- Superpipelined processors have longer instruction latency (in terms of cycles) than the SS processors, which can degrade performance in the presence of true dependencies
  - Note we're improving throughput at the expense of latency!
- Superscalar processors are more susceptible to resource conflicts—but we can fix this with hardware!

Instruction vs. Machine Parallelism

- Instruction-level parallelism (ILP) of a program—a measure of the average number of instructions in a ***program*** that, in theory, a processor might be able to execute at the same time
  - Mostly determined by the number of true (data) dependencies and procedural (control) dependencies in relation to the number of other instructions
  - ILP is traditionally “extracting parallelism from a single instruction stream working on a single stream of data”

Instruction vs. Machine Parallelism

- Machine parallelism of a processor—a measure of the ability of the ***processor*** to take advantage of the ILP of the program
  - Determined by the number of instructions that can be fetched and executed at the same time
  - A perfect machine with infinite machine parallelism can achieve the ILP of a program
- ***To achieve high performance, need both ILP and machine parallelism***

Matrix Multiplication

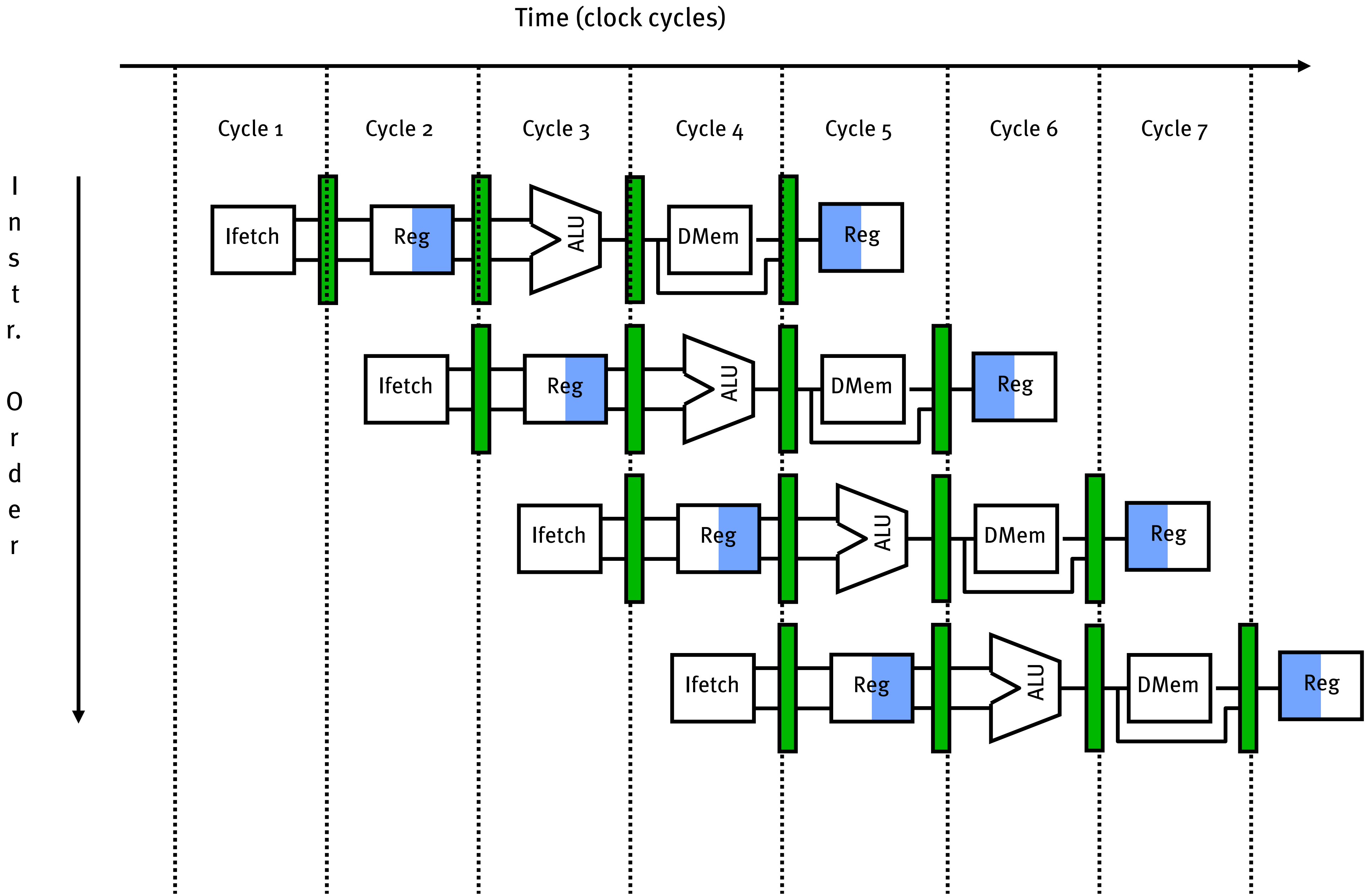
$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Assembly for y0

- `y0 = m00*x0 + m01*x1 + m02*x2 + m03*x3`

-
  ```
  t0 = m00 * x0
  t1 = m01 * x1
  t2 = m02 * x2
  t3 = m03 * x3
  t4 = t0 + t1
  t5 = t2 + t3
  y0 = t4 + t5
  ```

Pipelined Processor



Review: Pipeline Hazards

- Structural hazards
  - What are they?
  - How do we eliminate them?

Review: Pipeline Hazards

- Data hazards—read after write
  - What are they?
  - How do we eliminate them?

Review: Pipeline Hazards

- Control hazards—beq, bne, j, jr, jal
  - What are they?
  - How do we eliminate them?

Hazards are bad because they reduce the amount of achievable machine parallelism and keep us from achieving all the ILP in the instruction stream.

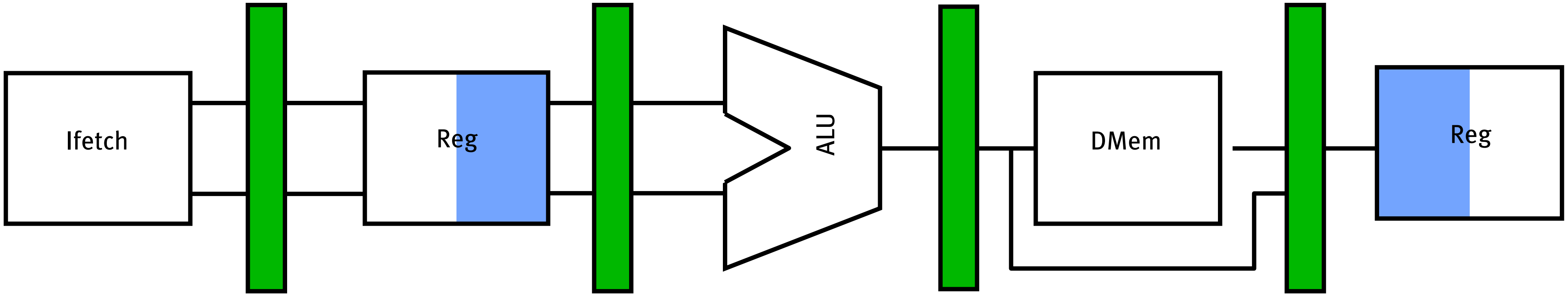
Machine Parallelism

- There are 2 main approaches for machine parallelism. Responsibility of resolving hazards is …
    - Primarily hardware-based—“dynamic issue”, “superscalar”
        - Today’s topic
    - Primarily software-based—“VLIW”

Multiple-Issue Processor Styles

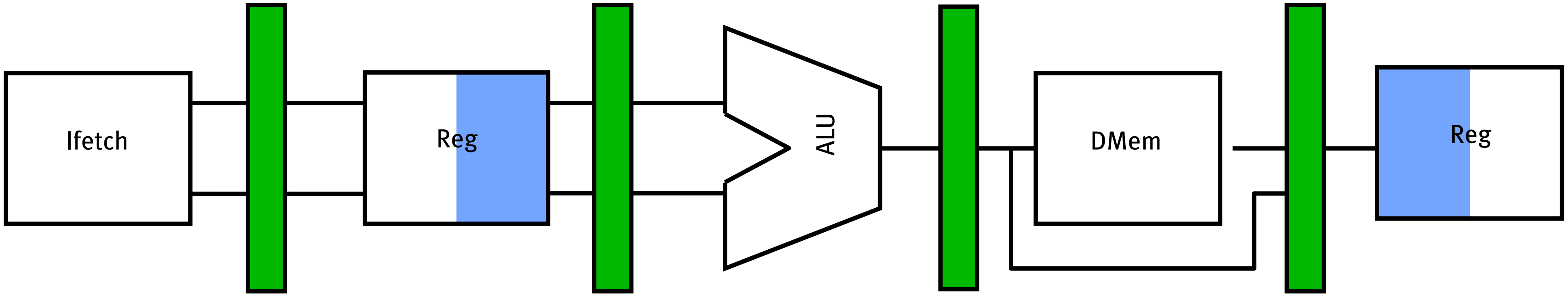
- Dynamic multiple-issue processors (aka superscalar)
  - Decisions on which instructions to execute simultaneously are being made dynamically (at run time by the hardware)
    - E.g., IBM Power 2, Pentium Pro/2/3/4, Core, MIPS R10K, HP PA 8500
    - We’re talking about this today
- Static multiple-issue processors (aka VLIW)
  - Decisions on which instructions to execute simultaneously are being made statically (at compile time by the compiler)
    - E.g., Intel Itanium and Itanium 2 for the IA-64 ISA—EPIC (Explicit Parallel Instruction Computer)
    - We’ll talk about this later

How do we support machine parallelism?



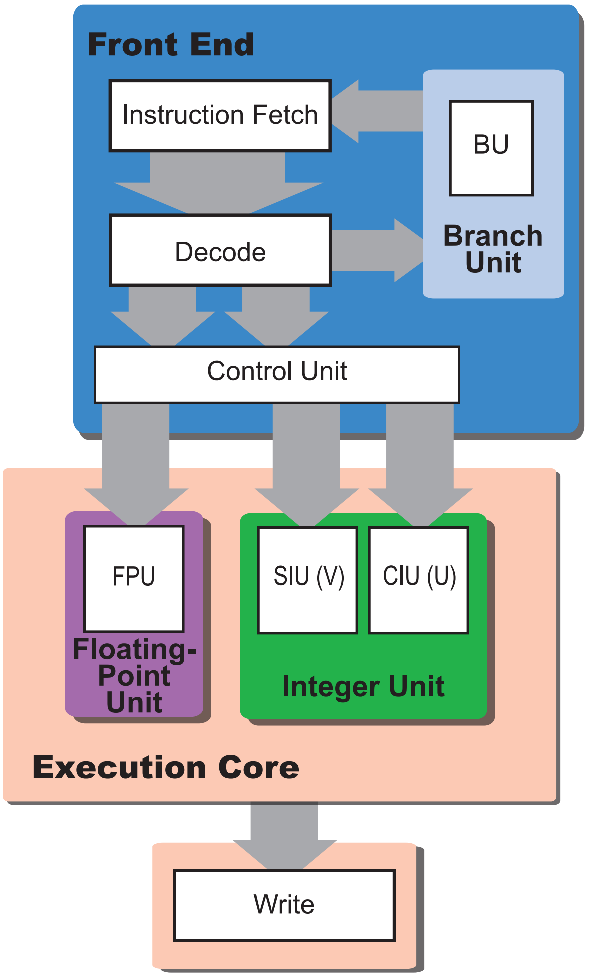
- First, let’s support parallel integer & FP instructions (MIPS)

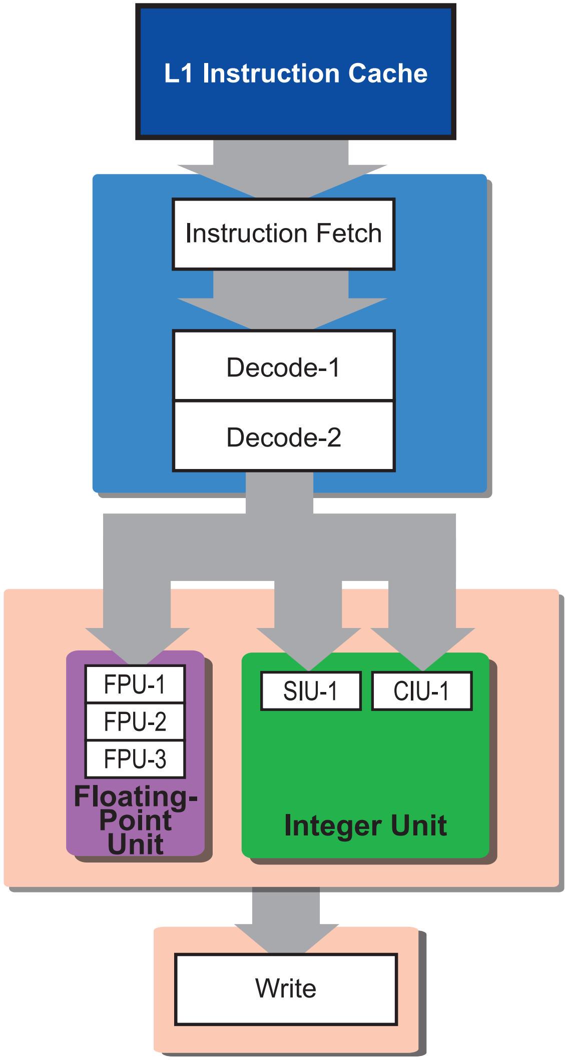
How do we support machine parallelism?



- Now, how do we support multiple integer instructions?

Pentium Microarchitecture





Pentium issue restrictions

- What restrictions would we need to place on the instructions in the U and V pipes to ensure correct execution?

Additional restrictions

- Some instructions are not pairable
  - some shift/rotate, long arith., extended, some fp, etc.
- Some instructions can only be issued to U
  - carry/borrow, prefix, shift w/ immediate, some fp
- Both instructions access same d-cache memory bank
- Multi-cycle instructions that write to memory must stall second pipe until last write

Multiple-Issue Datapath Responsibilities

- Must handle, with a combination of hardware and software fixes, the fundamental limitations of
  - Storage (data) dependencies—aka data hazards
    - Most instruction streams do not have huge ILP so …
    - … this limits performance in a superscalar processor

Multiple-Issue Datapath Responsibilities

- Must handle, with a combination of hardware and software fixes, the fundamental limitations of
  - Procedural dependencies—aka control hazards
    - Ditto, but even more severe
    - Use dynamic branch prediction to help resolve the ILP issue
      - Future lecture

Multiple-Issue Datapath Responsibilities

- Must handle, with a combination of hardware and software fixes, the fundamental limitations of
    - Resource conflicts—aka structural hazards
        - A SS/VLIW processor has a much larger number of potential resource conflicts
        - Functional units may have to arbitrate for result buses and register-file write ports
        - Resource conflicts can be eliminated by duplicating the resource or by pipelining the resource

Instruction Issue and Completion Policies

- Instruction-issue—initiate execution
    - Instruction lookahead capability—fetch, decode and issue instructions beyond the current instruction
- Instruction-completion—complete execution
    - Processor lookahead capability—complete issued instructions beyond the current instruction
- Instruction-commit—write back results to the RegFile or D$ (i.e., change the machine state)

In-order issue with in-order completion
In-order issue with out-of-order completion
Out-of-order issue with out-of-order completion and in-order commit
Out-of-order issue with out-of-order completion

In-Order Issue with In-Order Completion

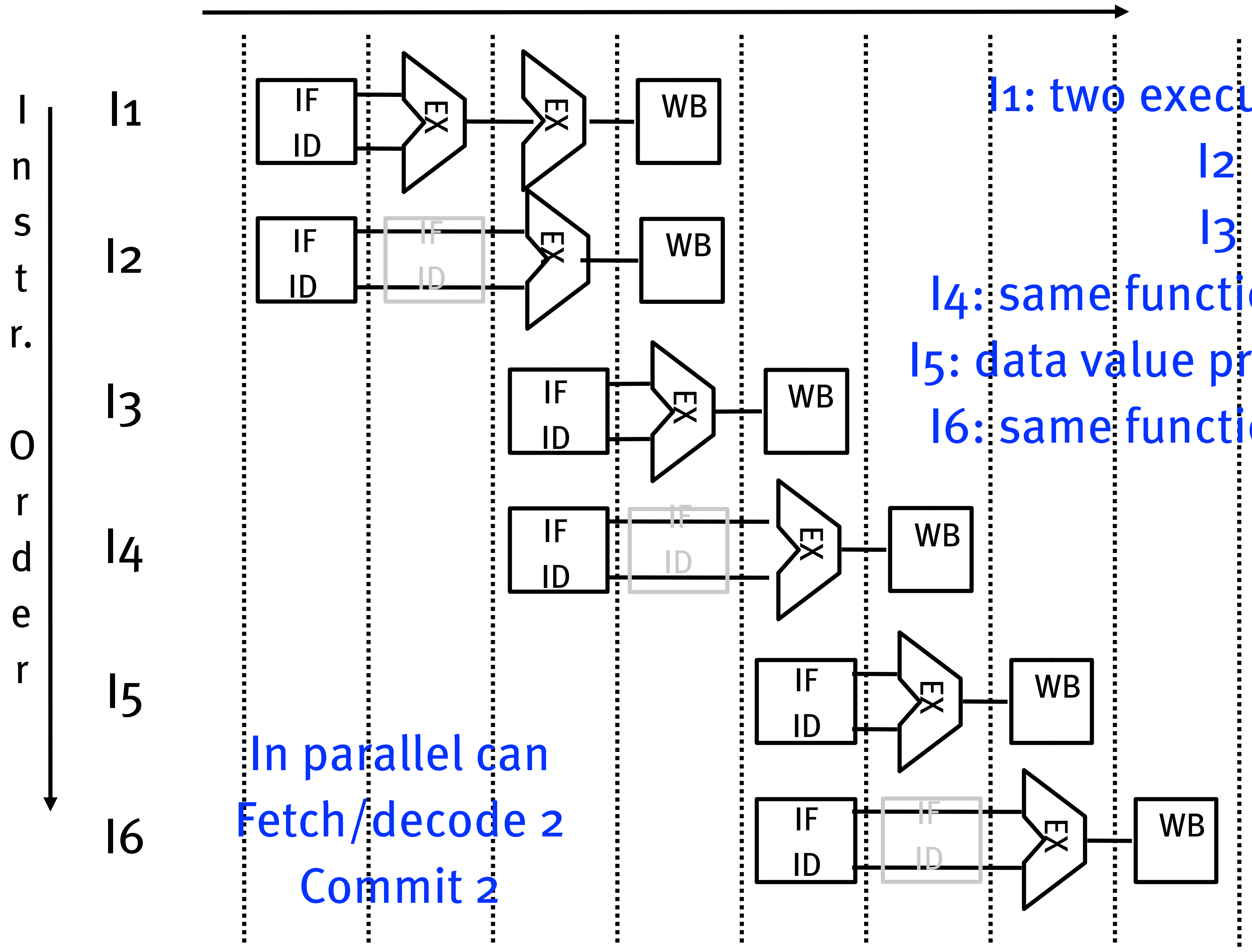
- Simplest policy is to issue instructions in exact program order and to complete them in the same order they were fetched (i.e., in program order)

In-Order Issue with In-Order Completion (Ex.)

- Assume a pipelined processor that can fetch and decode two instructions per cycle, that has three functional units (a single cycle adder, a single cycle shifter, and a two cycle multiplier), and that can complete (and write back) two results per cycle

- Instruction sequence:
  I1 – needs two execute cycles (a multiply)
  I2
  I3
  I4 – needs the same function unit as I3
  I5 – needs data value produced by I4
  I6 – needs the same function unit as I5

In-Order Issue, In-Order Completion Example



I1: two execute cycles

I2

I3

I4: same function unit as I3

I5: data value produced by I4

I6: same function unit as I5

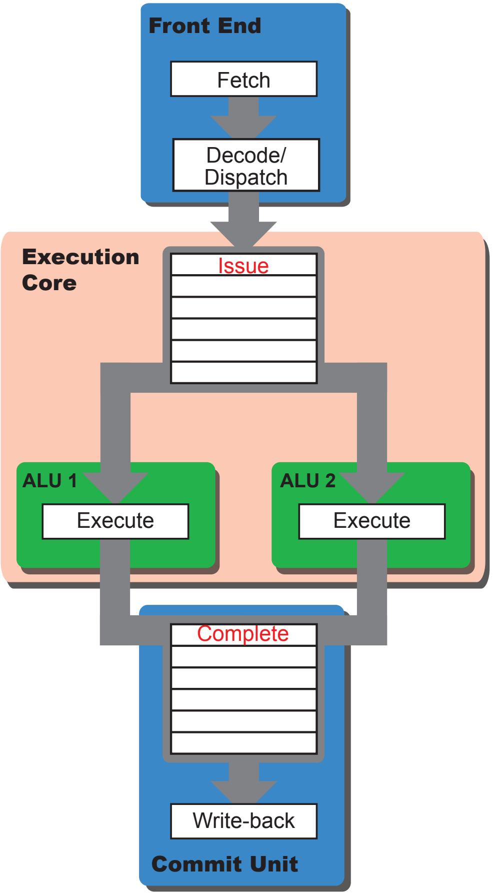
In parallel can

Fetch/decode 2

Commit 2

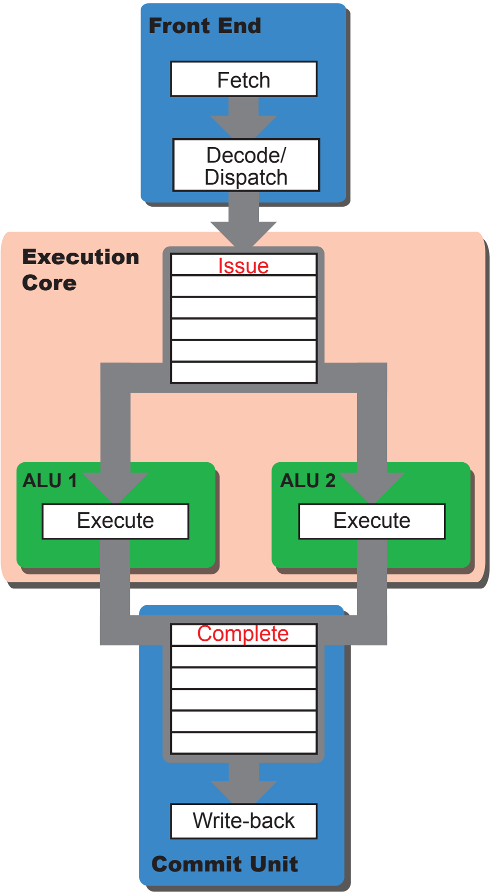
Pentium Retrospective

- Limited in performance by “front end”
  - Has to support variable-length instrs and segments
- Supporting all x86 features tough!
  - 30% of transistors are for legacy support
    - Up to 40% in Pentium Pro!
    - Down to 10% in P4
  - Microcode ROM is huge



Pentium Retrospective

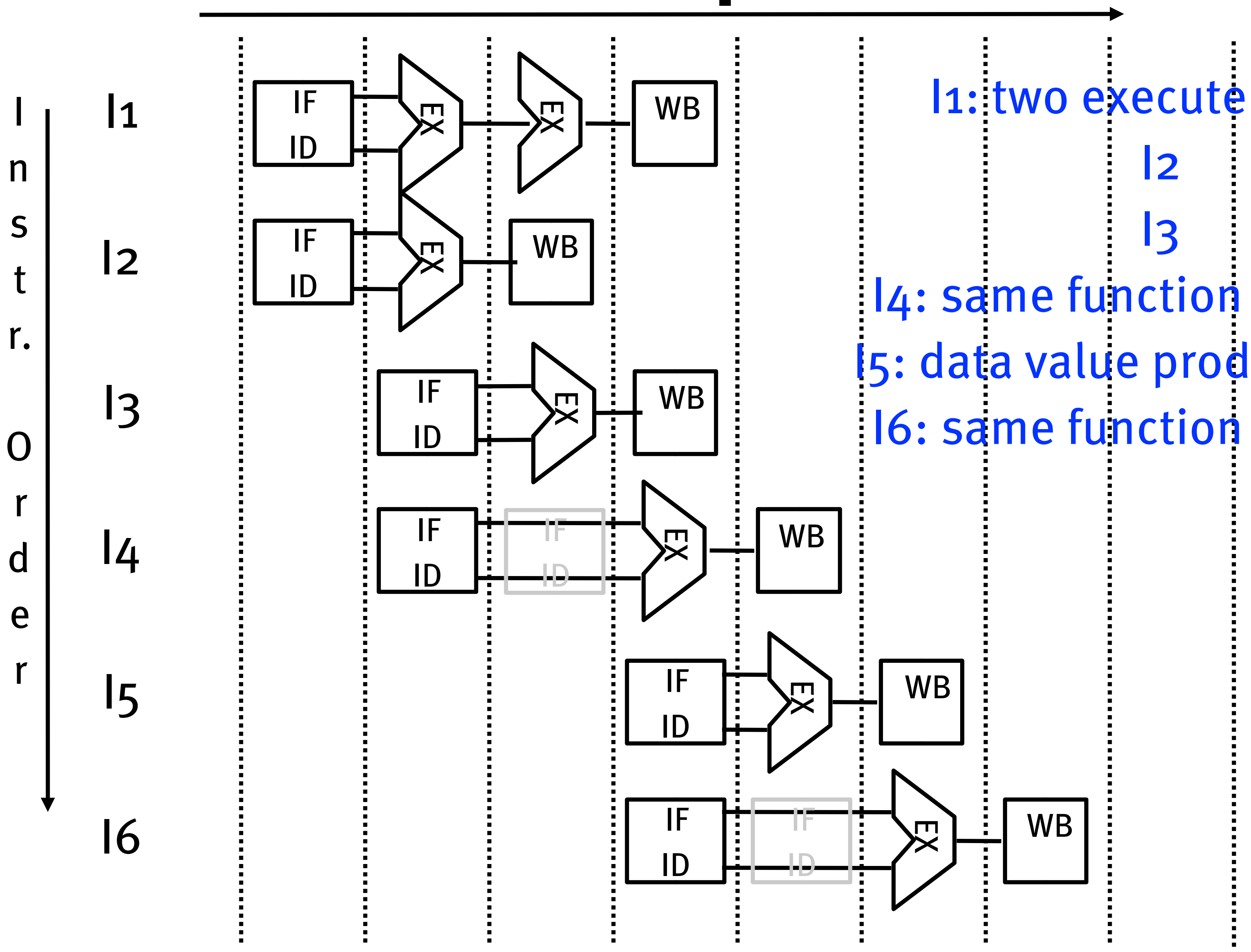
- Pentium is in-order issue, in-order complete
- “Static scheduling” by the dispatch logic:
  - Fetch/dispatch/execute/retire: all in order
- Drawbacks:
  - Adapts poorly to dynamic code stream
  - Adapts poorly to future hardware
    - What if we had 3 pipes not 2?



In-Order Issue with Out-of-Order Completion

- With out-of-order completion, a later instruction may complete before a previous instruction
  - Out-of-order completion is used in single-issue pipelined processors to improve the performance of long-latency operations such as divide
- When using out-of-order completion instruction issue is stalled when there is a resource conflict (e.g., for a functional unit) or when the instructions ready to issue need a result that has not yet been computed

IOI-OOC Example



I1: two execute cycles

I2

I3

I4: same function unit as I3

I5: data value produced by I4

I6: same function unit as I5

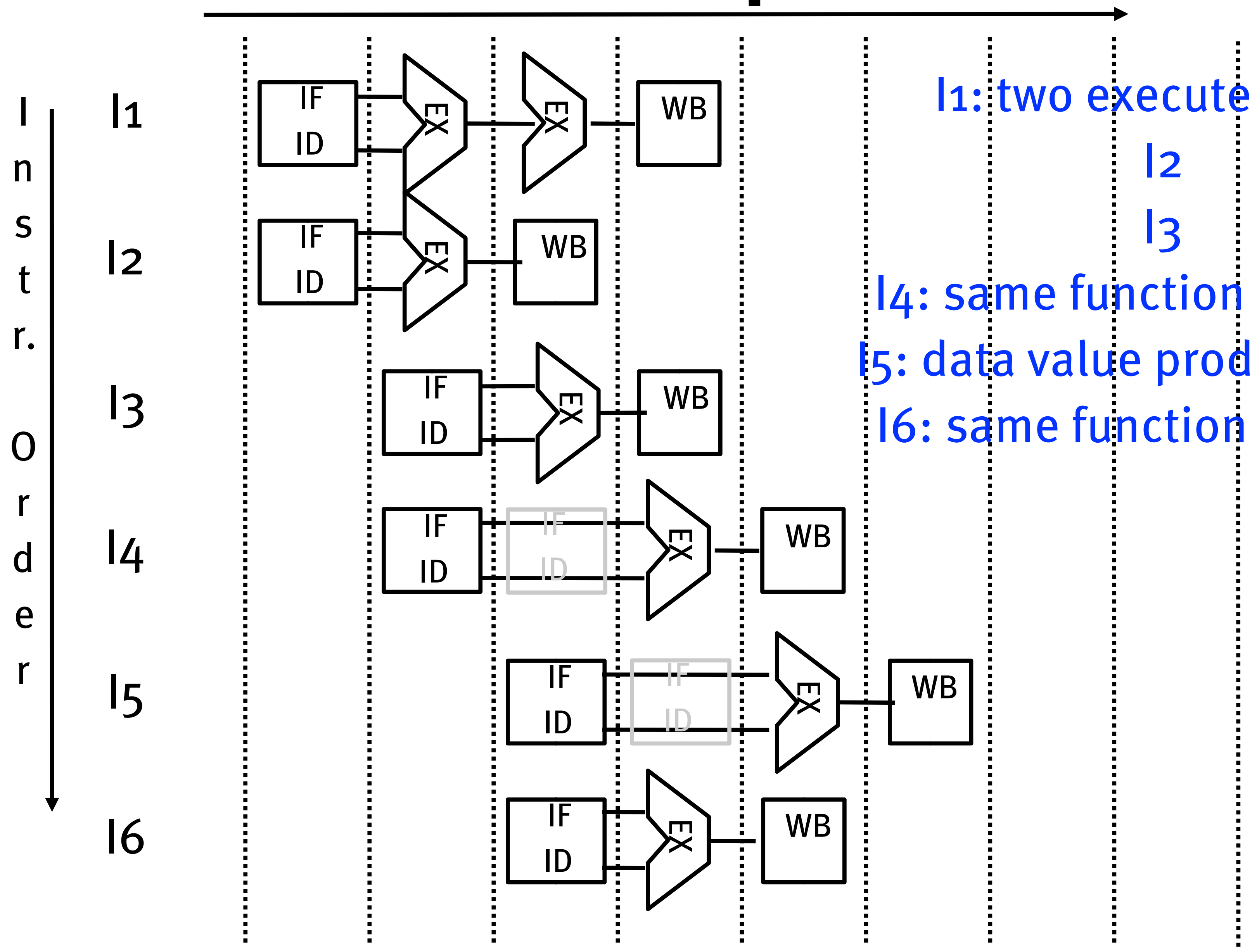
Handling Output Dependencies

- There is one more situation that stalls instruction issuing with IOI-OOC, assume
  - I1 – writes to R3
    I2 – writes to R3
    I5 – reads R3
- If the I1 write occurs after the I2 write, then I5 reads an incorrect value for R3
- I2 has an output dependency on I1—write before write
- The issuing of I2 would have to be stalled if its result might later be overwritten by an previous instruction (i.e., I1) that takes longer to complete—the stall happens before instruction issue
- While IOI-OOC yields higher performance, it requires more dependency checking hardware (both write-after-read and write-after-write)

Out-of-Order Issue with Out-of-Order Completion

- With in-order issue the processor stops decoding instructions whenever a decoded instruction has a resource conflict or a data dependency on an issued, but uncompleted instruction
  - The processor is not able to look beyond the conflicted instruction even though more downstream instructions might have no conflicts and thus be issueable
- Fetch and decode instructions beyond the conflicted one ("instruction window": Tetris), store them in an instruction buffer (as long as there's room), and flag those instructions in the buffer that don't have resource conflicts or data dependencies
- Flagged instructions are then issued from the buffer without regard to their program order

OOI-OOC Example



I1: two execute cycles

I2

I3

I4: same function unit as I3

I5: data value produced by I4

I6: same function unit as I5

Dependency Examples

-
  ```
  R3 := R3 * R5
  R4 := R3 + 1
  R3 := R5 + 1
  ```

True data dependency (RAW)

Output dependency (WAW)

Antidependency (WAR)

Antidependencies (WAR)

- With OOI also have to deal with data antidependencies – when a later instruction (that completes earlier) produces a data value that destroys a data value used as a source in an earlier instruction (that issues later)

- The constraint is similar to that of true data dependencies, except reversed
    - Instead of the later instruction using a value (not yet) produced by an earlier instruction (read before write), the later instruction produces a value that destroys a value that the earlier instruction (has not yet) used (write before read)

Dependencies Review

- Each of the three data dependencies …
  - True data dependencies (read before write)
  - Antidependencies (write before read)
  - Output dependencies (write before write)

  storage conflicts
- … manifests itself through the use of registers (or other storage locations)
- True dependencies represent the flow of data and information through a program
- Anti- and output dependencies arise because the limited number of registers mean that programmers reuse registers for different computations
- When instructions are issued out-of-order, the correspondence between registers and values breaks down and the values conflict for registers

Conflict example

- (1) R3 := R3 * R5
  (2) R4 := R3 + 1
  (3) R3 := R5 + 1

  - (3) must ensure that (1) and (2) read R3 before (3) writes it

  - But the value in (3)'s R3 has nothing to do with the value in (1) and (2)'s R3! Shouldn't we be able to issue that instruction?

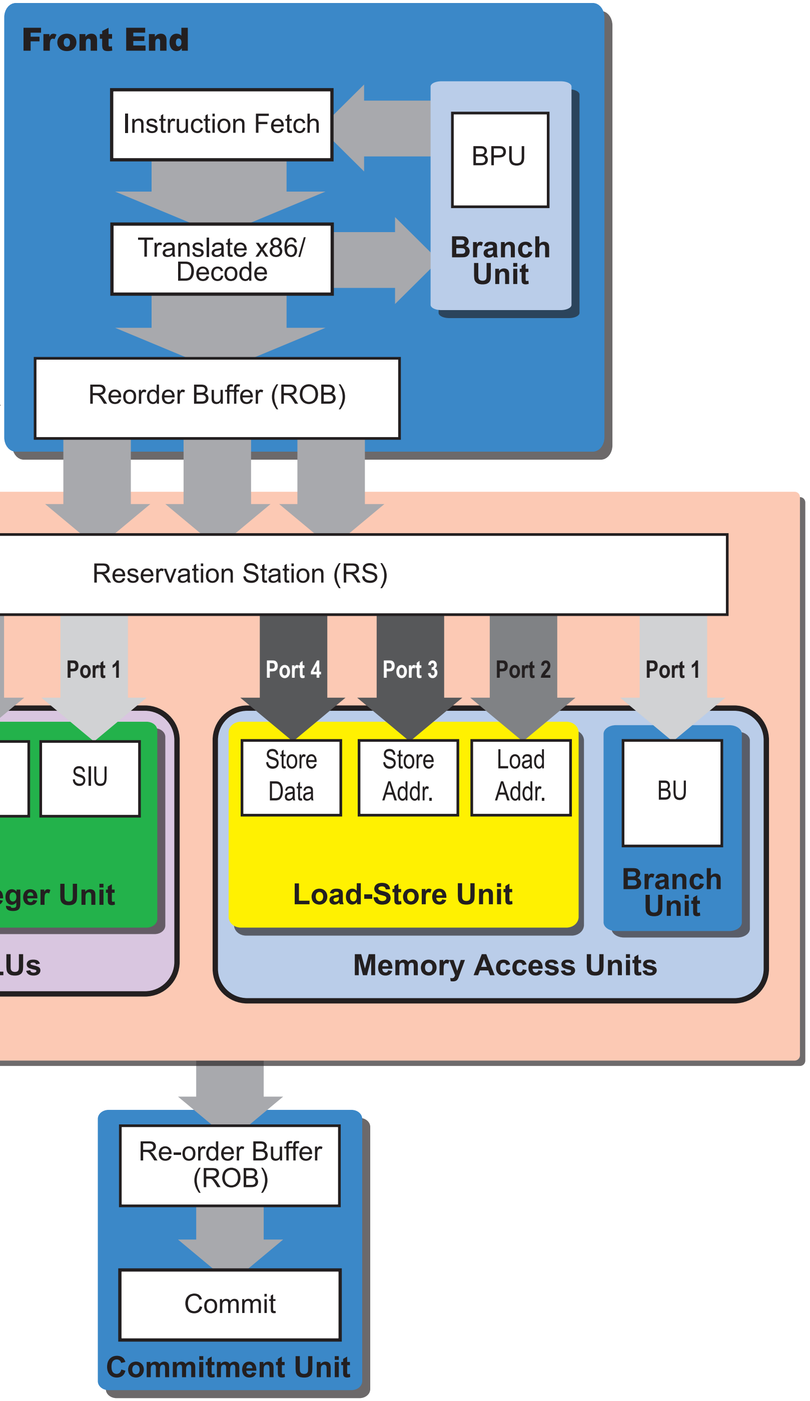
Storage Conflicts and Register Renaming

- Storage conflicts can be reduced (or eliminated) by increasing or duplicating the troublesome resource
    - Provide additional registers that are used to reestablish the correspondence between registers and values
        - Allocated dynamically by the hardware in SS processors
- Register renaming — the processor renames the original register identifier in the instruction to a new register (one not in the visible register set)

    -
      ```
      R3 := R3 * R5        R3b := R3a * R5a
      R4 := R3 + 1         R4a := R3b + 1
      R3 := R5 + 1         R3c := R5a + 1
      ```
- The hardware that does renaming assigns a “replacement” register from a pool of free registers and releases it back to the pool when its value is superseded and there are no outstanding references to it   [future lecture!]

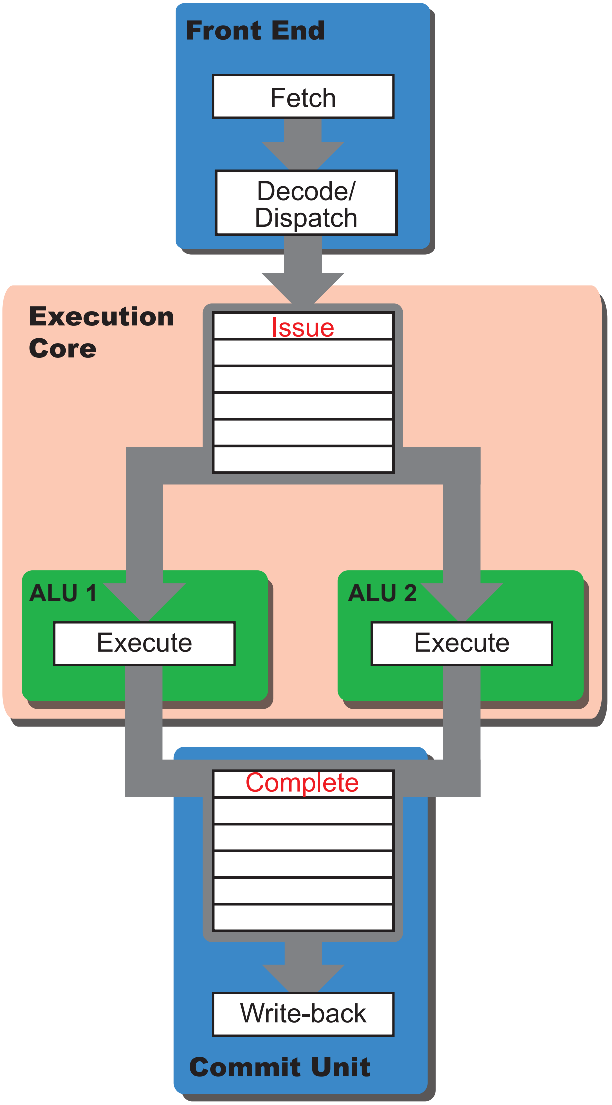
Pentium Pro

register renaming happens here ->



Pentium Pro

1. Fetch	In order
2. Decode/dispatch	In order
3. Issue	Reorder
4. Execute	Out of order
5. Complete	Reorder
6. Writeback (commit)	In order

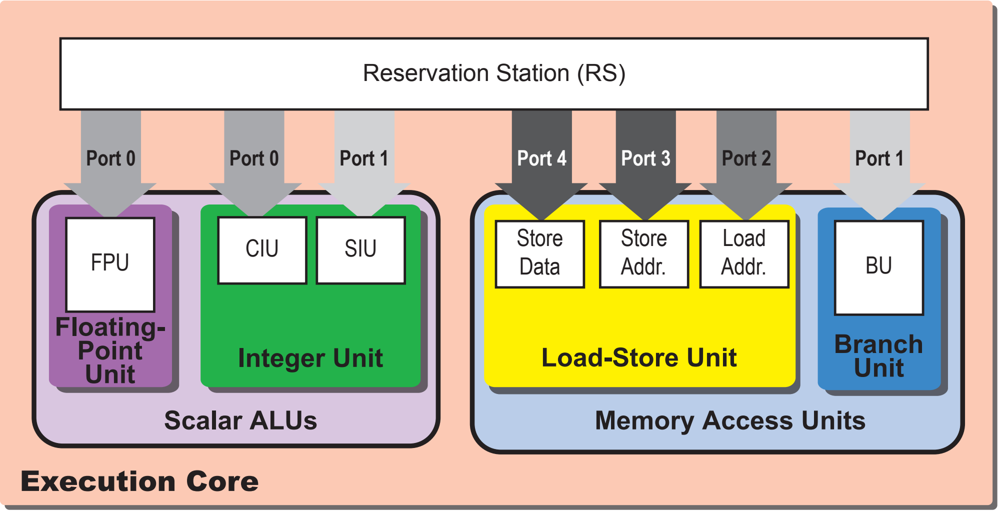


P6 Pipeline

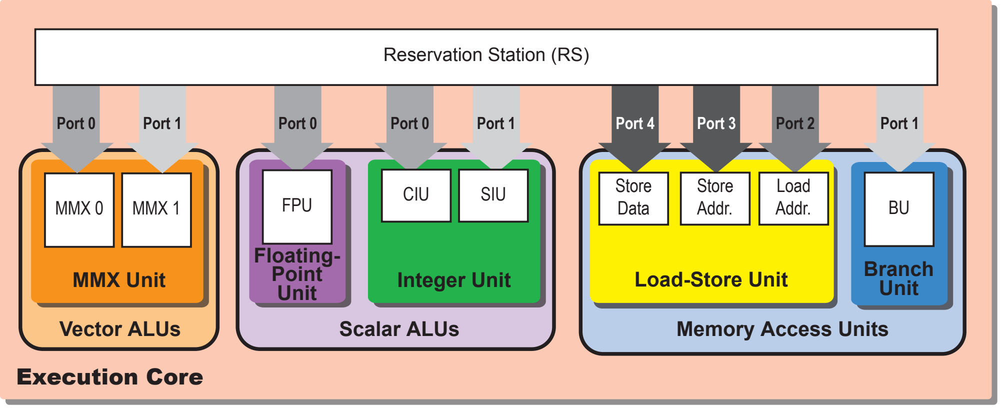
- Instruction fetch, BTB access (3.5 stages)
  - 2 cycles for instruction fetch
- Decode, x86->uops (2.5 stages)
- Register rename (1 stage)
- Write to reservation station (1 stage)
- Read from reservation station (1 stage)
- Execute (1+ stages)
- Commit (2 stages)

Pentium Pro backends

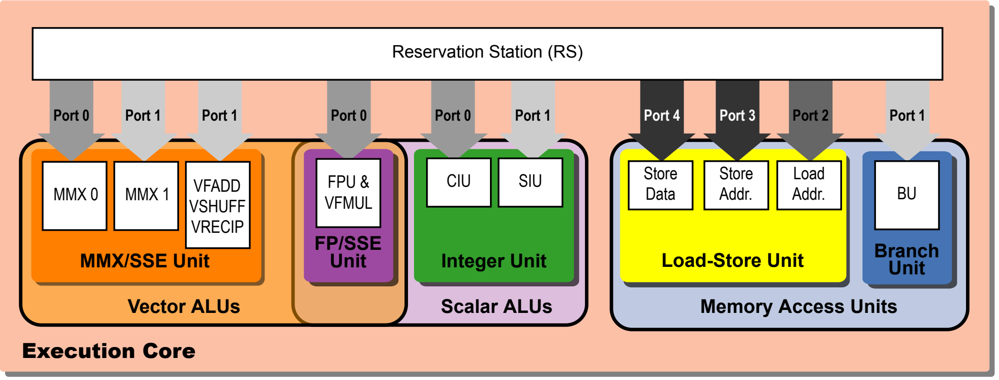
- Pentium Pro



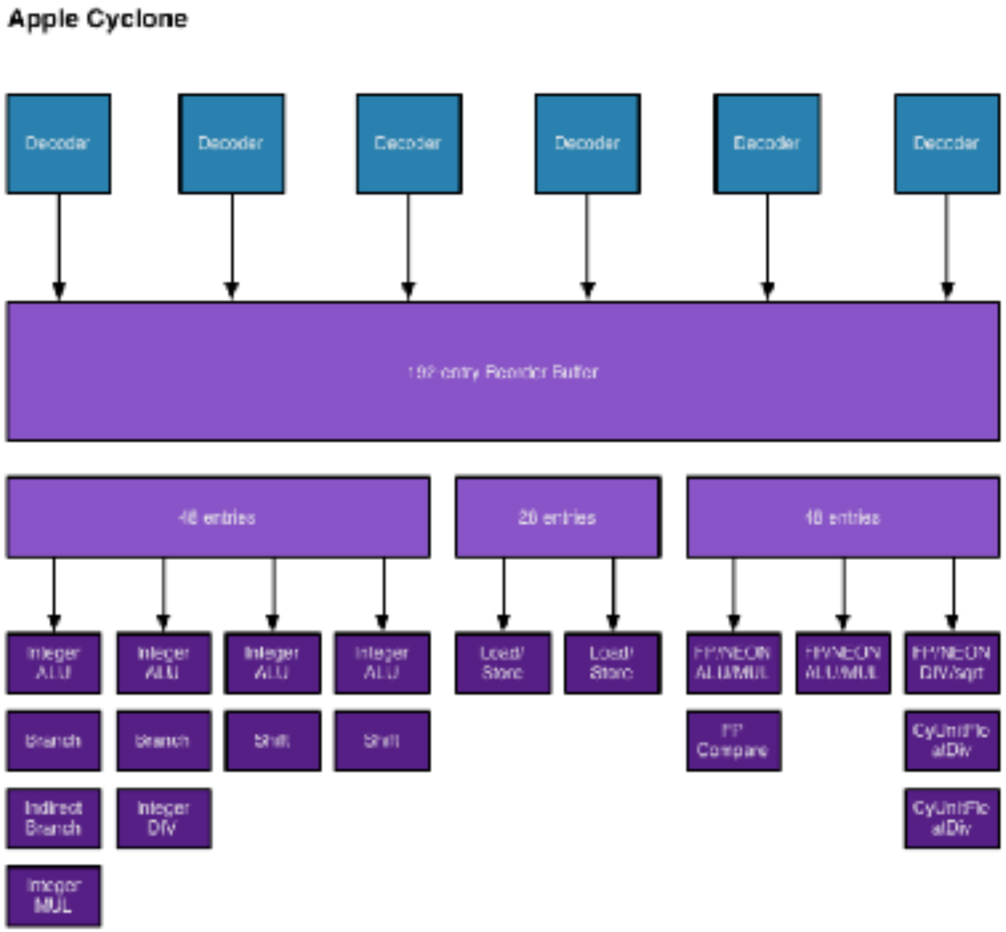
- Pentium 2

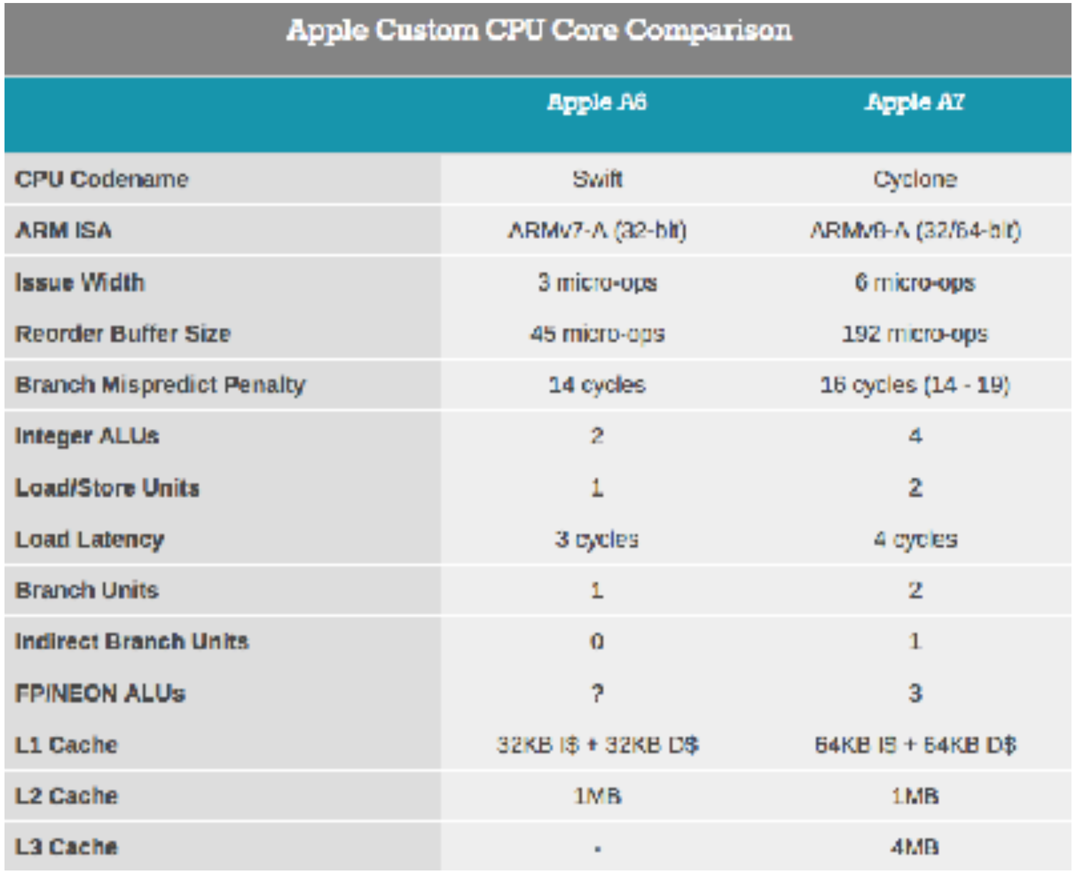


- Pentium 3



Apple “Cyclone” A7 (2014)



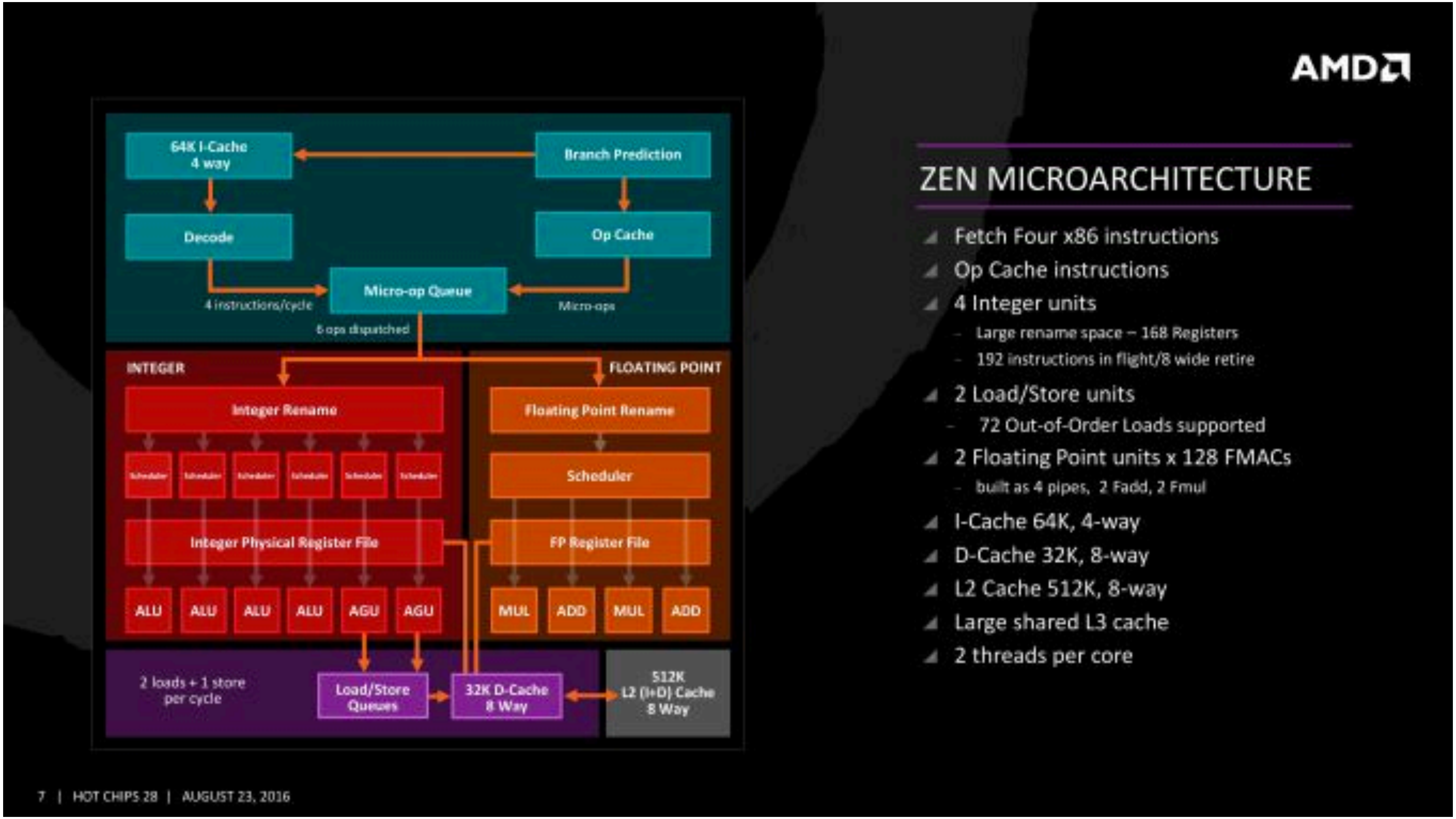


Apple Custom CPU Core Comparison

	Apple A6	Apple A7
CPU Codename	Swift	Cyclone
ARM ISA	ARMv7-A (32-bit)	ARMv8-A (32/64-bit)
Issue Width	3 micro-ops	6 micro-ops
Reorder Buffer Size	45 micro-ops	192 micro-ops
Branch Mispredict Penalty	14 cycles	16 cycles (14 - 19)
Integer ALUs	2	4
Load/Store Units	1	2
Load Latency	3 cycles	4 cycles
Branch Units	1	2
Indirect Branch Units	0	1
FP/NEON ALUs	?	3
L1 Cache	32KB I$ + 32KB D$	64KB I$ + 64KB D$
L2 Cache	1MB	1MB
L3 Cache	-	4MB

http://www.anandtech.com/show/7910/apples-cyclone-microarchitecture-detailed

AMD Zen (2016)



ZEN MICROARCHITECTURE

- Fetch Four x86 instructions
- Op Cache instructions
- 4 Integer units
  - Large rename space – 168 Registers
  - 192 instructions in flight/8 wide retire
- 2 Load/Store units
  - 72 Out-of-Order Loads supported
- 2 Floating Point units x 128 FMACs
  - built as 4 pipes, 2 Fadd, 2 Fmul
- I-Cache 64K, 4-way
- D-Cache 32K, 8-way
- L2 Cache 512K, 8-way
- Large shared L3 cache
- 2 threads per core