

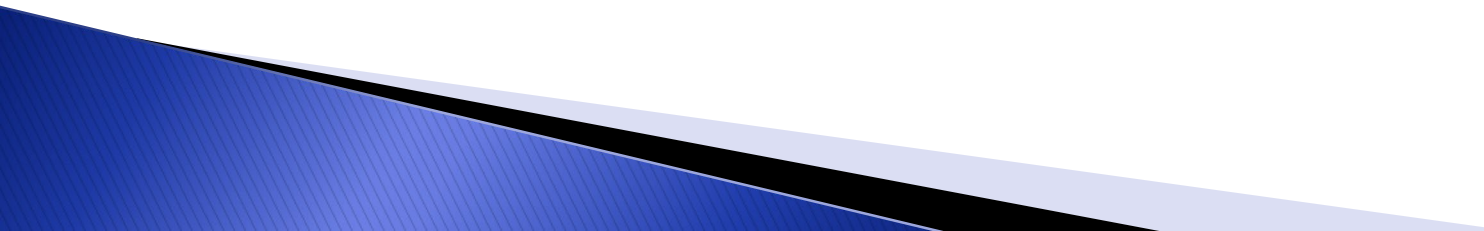
CSE 2421

CISC / RISC

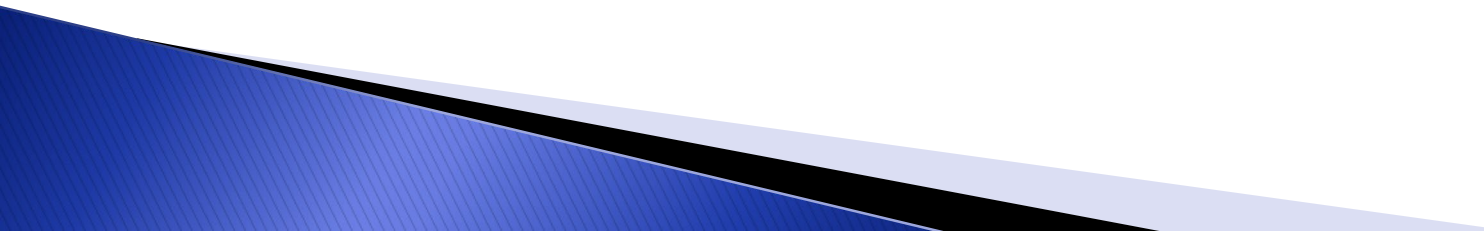
Computer Architectures

Required reading: The Aside in section 4.1, pg 361–363
5.7 and 5.7.1

Prelude – restartable instructions

- ▶ Any instruction that touches memory could generate a page fault
 - ▶ Instructions that page fault are rolled back and then restarted when the desired page of memory is brought in
 - ▶ This mandates that all memory instructions keep track of everything they are doing so that they can rollback those internal changes to allow a restart
- 

Complex Instruction Set Computers

- ▶ Do more with each instruction: “loop” “leave”
 - ▶ Tolerate a slower clock rate
 - ▶ Tolerate a complex micro-architecture underneath the visible architecture
 - ▶ Hard to minimize power with that complex micro-architecture
 - ▶ Supposed to be closer to how people think / closer to high level programming languages
 - ▶ Designs predate cache memory
- 

Reduced Instruction Set Computers

- ▶ Let nothing get in the way of a faster clock rate
- ▶ More simple instructions per second beats fewer complex instructions per second because many useful instructions are simple
- ▶ The compiler is good enough to bridge the very low level machine instructions to the higher level language, and a simple machine is easier to write effective compilers for
- ▶ Easier to make low power / small footprint
- ▶ Cache and pipeline
- ▶ Fixed length instructions with simple encoding

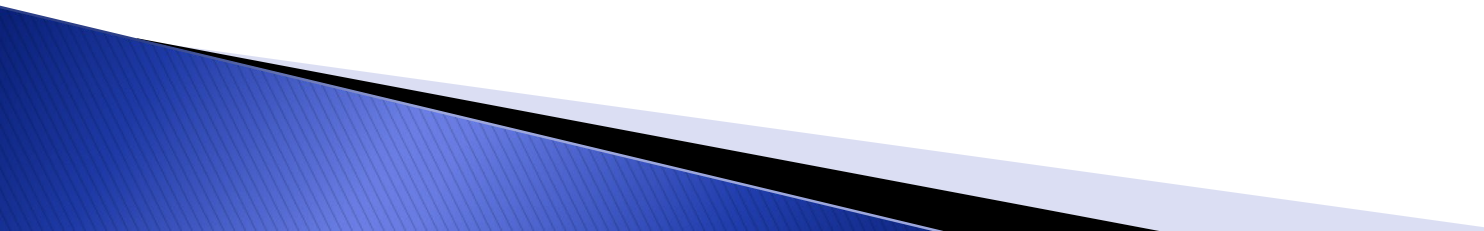
RISC history / design decisions

- ▶ More registers!
- ▶ Load/store architecture
- ▶ Fixed length instructions
- ▶ No status flags on early designs

Why?



More registers

- ▶ Register ops are the fastest possible ops (the L1 cache on intel is 4 clocks ticks away, memory is 230+ clocks)
 - ▶ Without complex addressing modes, we need more pointer and index registers to properly access memory
 - ▶ Enough registers means minimal memory accesses
 - ▶ Too many means expensive context switching
- 

Load / Store architecture

- ▶ No direct memory ALU operations
 - ~~◦ `add %rax, (%rdi)`~~
- ▶ Simple loads and stores take 2 registers:
 - Pointer register to provide the address
 - Data register to write or to read into
- `movq %rax, (%rdi)` # OK in RISC
 - ~~◦ `movq disp(%rdi, %rsi, 8), %rax` #too complicated! (or is it?)~~

Fixed-length instructions

- ▶ 32 bit instructions for ARM in normal mode
- ▶ 8 to 120 bit long instructions for x86-64
- ▶ Fixed-length means pipeline friendly
 - You don't have to read the first byte to know how many bytes to fetch
 - You don't need variable width data paths to read "an instruction"
 - You don't need a variable width instruction decoder

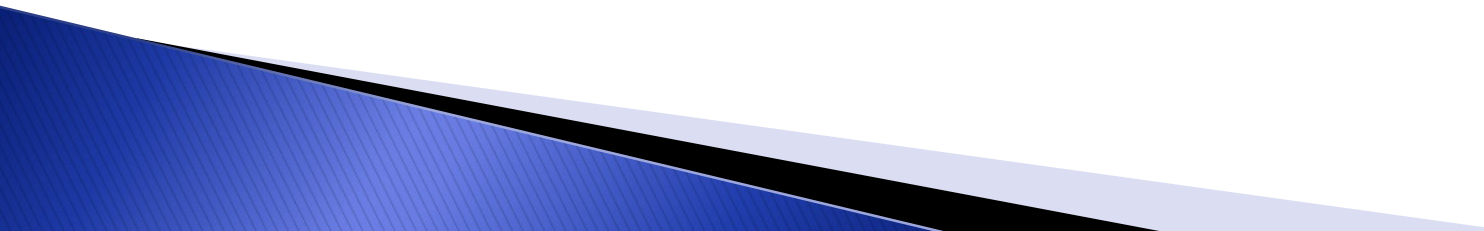
No automatic status flags on early designs

- ▶ Flags are often ignored by the next instruction, which often sets the flags to something else
- ▶ Flags delay the settling of *all* ALU operations
 - Wait for the ALU op to complete (unavoidable)
 - Wait for the flag logic to settle (small but non-zero)
 - Write the flag register (small but non-zero)
- ▶ Create hazards in out-of-order, register renamed designs
 - Means many renamed copies of those flag bits internally
 - Means dependency checks with those bits during speculative execution

ARM: “Advanced RISC Machine”

- ▶ RISC design inspired by Berkeley RISC project
- ▶ Low-power
- ▶ Highly flexible – the architecture is extensible
- ▶ Conditional execution on nearly all instructions
- ▶ Shifts can be included as part of some operations
- ▶ 32 bit ARM has 16 general purpose registers
- ▶ 64 bit ARM has 32 general purpose registers

If it is not running an intel CPU, it is almost always an ARM CPU.



Apple A12 (based on ARMv8.3) 2018

- ▶ 7nm fabrication
- ▶ 2 high-performance cores + 4 high efficiency cores
- ▶ 128 KB instruction + 128 KB data L1 caches
- ▶ 8 MB L2 cache
- ▶ Supports out-of-order execution, branch prediction
- ▶ 4 core on-board GPU + 8 core neural network engine

Per-core:

- ▶ 31 general purpose 64 bit registers (2x intel)
- ▶ 16 instruction pipeline depth

Aside from register count, all these numbers match peer intel

Apple M1 (based on ARMv8.3) 2020

- ▶ 5nm fabrication (intel was above 10, AMD was at 7nm)
- ▶ 2 high-performance cores
 - 128 KB instruction + 128 KB data L1 cache
 - 12 MB shared L2 cache
- ▶ 4 high-efficiency cores
 - 128 KB instruction + 64KB data L1 cache
 - 4 MB shared L2 cache
- ▶ Supports out-of-order execution, branch prediction
- ▶ 8 core on-board GPU + 16 core neural network engine + image processor

Per-core:

- ▶ 31 general purpose 64 bit registers (2x intel)
- ▶ 16 instruction pipeline depth

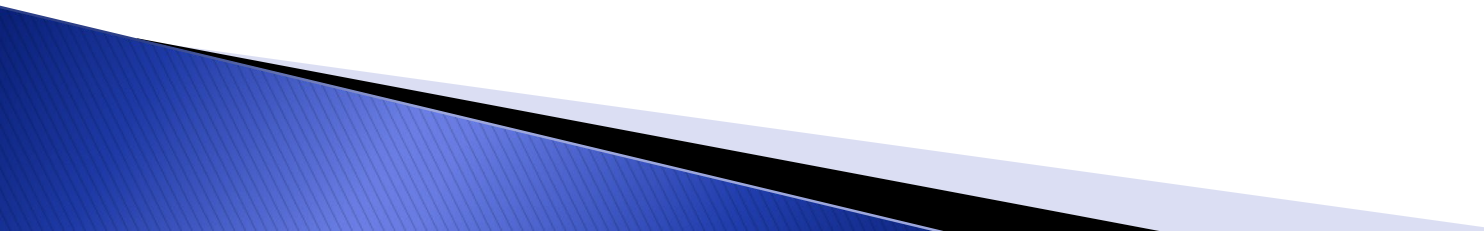
All of these numbers compare well current intel

Convergence

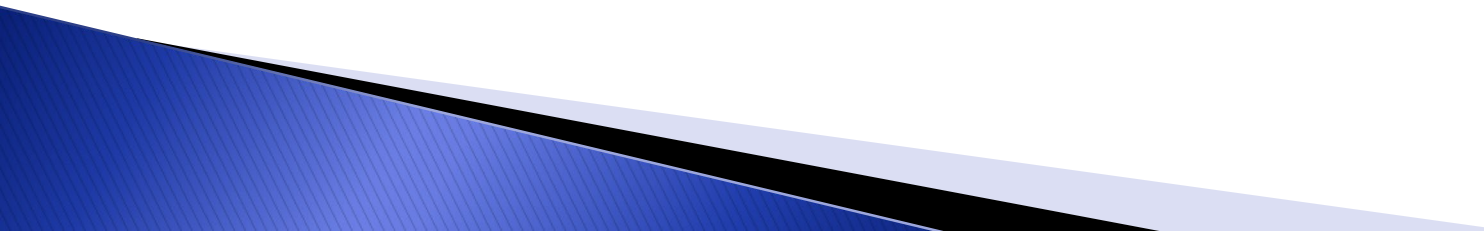
Both designs have valid ideas

- ▶ ARM has condition flags and 2-register addressing mode
- ▶ Intel has pipelining and successive chips have more and more single cycle instructions such as single cycle multiply

Pipelines and multi-cycle instructions

- ▶ A multi-cycle instruction stalls a simple single pipeline until the execution completes and the next instruction can begin execution
 - ▶ RISC solves this by having few multi-cycle instructions
 - ▶ CISC solves this with speculative execution and micro-ops
- 

Speculative execution

- ▶ Lets us use parallelism for increased throughput while maintaining a serial programming model
 - ▶ Don't bother unless you have parallel (underused) hardware
 - Modern intel has 4 integer ALU per core
 - ▶ Have to be ready to rollback the proposed state changes
 - ▶ Have to be able to infer dependencies between instructions
 - ▶ Has to be invisible to the programming model
 - ▶ Has side effects – notably the cache
 - ▶ Visible side effects can be security holes
- 

Wrap-up

- ▶ What began as 2 very different designs has converged
 - Underneath the CISC programming model, intel has many RISC features
 - There is such a thing as too simple – ARM has selectively included complex features
- ▶ There are concerns beyond pure high speed performance
 - Power consumption
 - Embedded base of existing customers resist incompatible changes
- ▶ “Purity” of design should serve a purpose
 - Intel needs to keep customers happy
 - ARM resistance to special cases breaks dependencies that hinder performance and security

Resources

- ▶ The convergence as seen in 1996: “Beyond RISC: The Post-RISC Architecture”

<https://www.cse.msu.edu/~enbody/postrisc/postrisc2.htm>

- ▶ Patterson & Hennessy, 2019 “A New Golden Age for Computer Architecture”

<https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext?mobile=false>

I strongly recommend reading the Patterson & Hennessy paper!



Resources

- ▶ The Wikipedia microarchitecture page:
<https://en.wikipedia.org/wiki/Microarchitecture>