Lessons from the ARM Architecture

Richard Grisenthwaite
Lead Architect and Fellow
ARM

ARM Processor Applications

Overview

- Introduction to the ARM architecture
  - Definition of “Architecture”
  - History & evolution
  - Key points of the basic architecture
- Examples of ARM implementations
  - Something to the micro-architects interested
- My Lessons on Architecture design
  - …or what I wish I’d known 15 years ago

Definition of “Architecture”

- The Architecture is the contract between the Hardware and the Software
  - Confers rights and responsibilities to both the Hardware and the Software
  - MUCH more than just the instruction set

- The architecture distinguishes between:
  - Architected behaviors:
    - Must be obeyed
    - May be just the limits of behavior rather than specific behaviors
  - Implementation specific behaviors – that expose the micro-architecture
    - Certain areas are declared implementation specific. E.g.:
      - Power-down
      - Cache and TLB Lockdown
      - Details of the Performance Monitors

- Code obeying the architected behaviors is portable across implementations
  - Reliance on implementation specific behaviors gives no such guarantee

- Architecture is different from Micro-architecture
  - What vs How

History

- ARM has quite a lot of history
  - First ARM core (ARM1) ran code in April 1985…
    - 3 stage pipeline very simple RISC-style processor
  - Original processor was designed for the Acorn Microcomputer
    - Replacing a 6502-based design
  - ARM Ltd formed in 1990 as an “Intellectual Property” company
    - Taking the 3 stage pipeline as the main building block
- This 3 stage pipeline evolved into the ARM7TDMI
  - Still the mainstay of ARM’s volume
  - Code compatibility with ARM7TDMI remains very important
    - Especially at the applications level
- The ARM architecture has features which derive from ARM1
  - Strong “applications level” compatibility focus in the ARM products

Evolution of the ARM Architecture

- Original ARM architecture:
  - 32-bit RISC architecture focussed on core instruction set
  - 16 Registers - 1 being the Program counter – generally accessible
  - Conditional execution on all instructions
  - Load/Store Multiple operations - Good for Code Density
  - Shifts available on data processing and address generation
  - Original architecture had 26-bit address space
    - Augmented by a 32-bit address space early in the evolution
- Thumb instruction set was the next big step
  - ARMv4T architecture (ARM7TDMI)
  - Introduced a 16-bit instruction set alongside the 32-bit instruction set
    - Different execution states for different instruction sets
    - Switching ISA as part of a branch or exception
    - Not a full instruction set – ARM still essential
  - ARMv4 architecture was still focused on the Core instruction set only

Evolution of the Architecture (2)

- **ARMv5TEJ (ARM926EJ-S) introduced:**
  - Better interworking between ARM and Thumb
    - Bottom bit of the address used to determine the ISA
  - DSP-focussed additional instructions
  - Jazelle-DBX for Java byte code interpretation in hardware
  - Some architecting of the virtual memory system
- ARMv6K (ARM1136JF-S) introduced:
  - Media processing – SIMD within the integer datapath
  - Enhanced exception handling
  - Overhaul of the memory system architecture to be fully architected
    - Supported only 1 level of cache
- ARMv7 rolled in a number of substantive changes:
  - Thumb-2* - variable length instruction set
  - TrustZone*
  - Jazelle-RCT
  - Neon

Extensions to ARMv7

- MPE – Multiprocessing Extensions
  - Added Cache and TLB Maintenance Broadcast for efficient MP
- VE - Virtualization Extensions
  - Adds hardware support for virtualization:
    - 2 stages of translation in the memory system
    - New mode and privilege level for holding an Hypervisor
      - With associated traps on many system relevant instructions
    - Support for interrupt virtualization
  - Combines with a System MMU
- LPAE – Large Physical Address Extensions
  - Adds ability to address up to 40-bits of physical address space

VFP – ARM's Floating-point solution

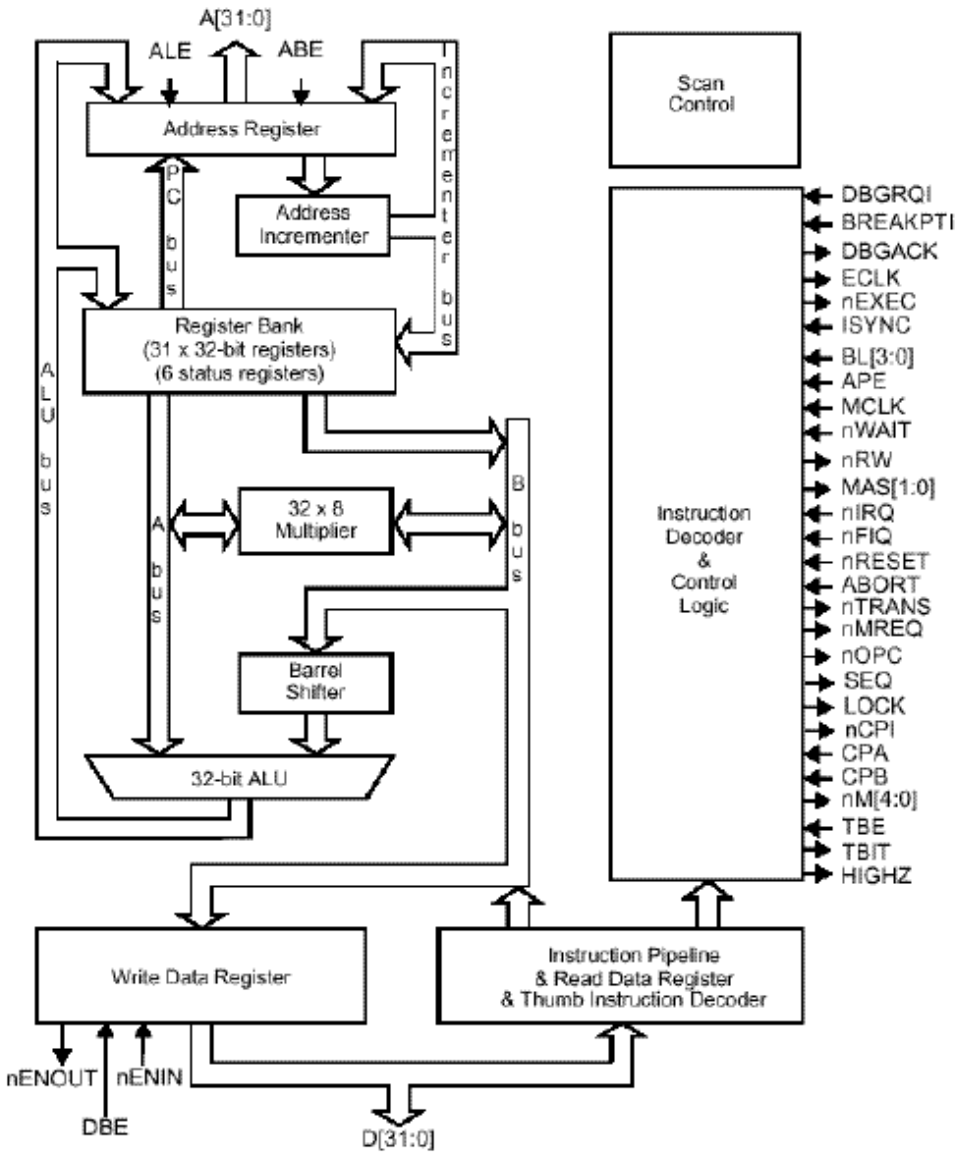
- VFP – "Vector Floating-point"
  - Vector functionality has been deprecated in favour of Neon
- Described as a "coprocessor"
  - Originally a tightly-coupled coprocessor
    - Executed instructions from ARM instruction stream via dedicated interface
  - Now more tightly integrated into the CPU
- Single and Double precision floating-point
  - Fully IEEE compliant
    - Until VFPv3, implementations required support code for denorms
  - Alternative Flush to Zero handling of denorms also supported
- Recent VFP versions:
  - VFPv3 – adding more DP registers (32 DP registers)
  - VFPv4 – adds Fused MAC and Half-precision support (IEEE754-2008)

ARM Architecture versions and products

- **Key architecture revisions and products:**
  - ARMv1-ARMv3: largely lost in the mists of time
  - ARMv4T: ARM7TDMI – first Thumb processor
  - ARMv5TEJ(+VFPv2): ARM926EJ-S
  - ARMv6K(+VFPv2): ARM1136JF-S, ARM1176JFZ-S, ARM11MPCore – first Multiprocessing Core
  - ARMv7-A+VFPv3 Cortex-A8
  - ARMv7-A+MPE+VFPv3: Cortex-A5, Cortex-A9
  - ARMv7-A+MPE+VE+LPAE+VFPv4 Cortex-A15
  - ARMv7-R : Cortex-R4, Cortex-R5
  - ARMv6-M Cortex–M0
  - ARMv7-M: Cortex-M3, Cortex-M4

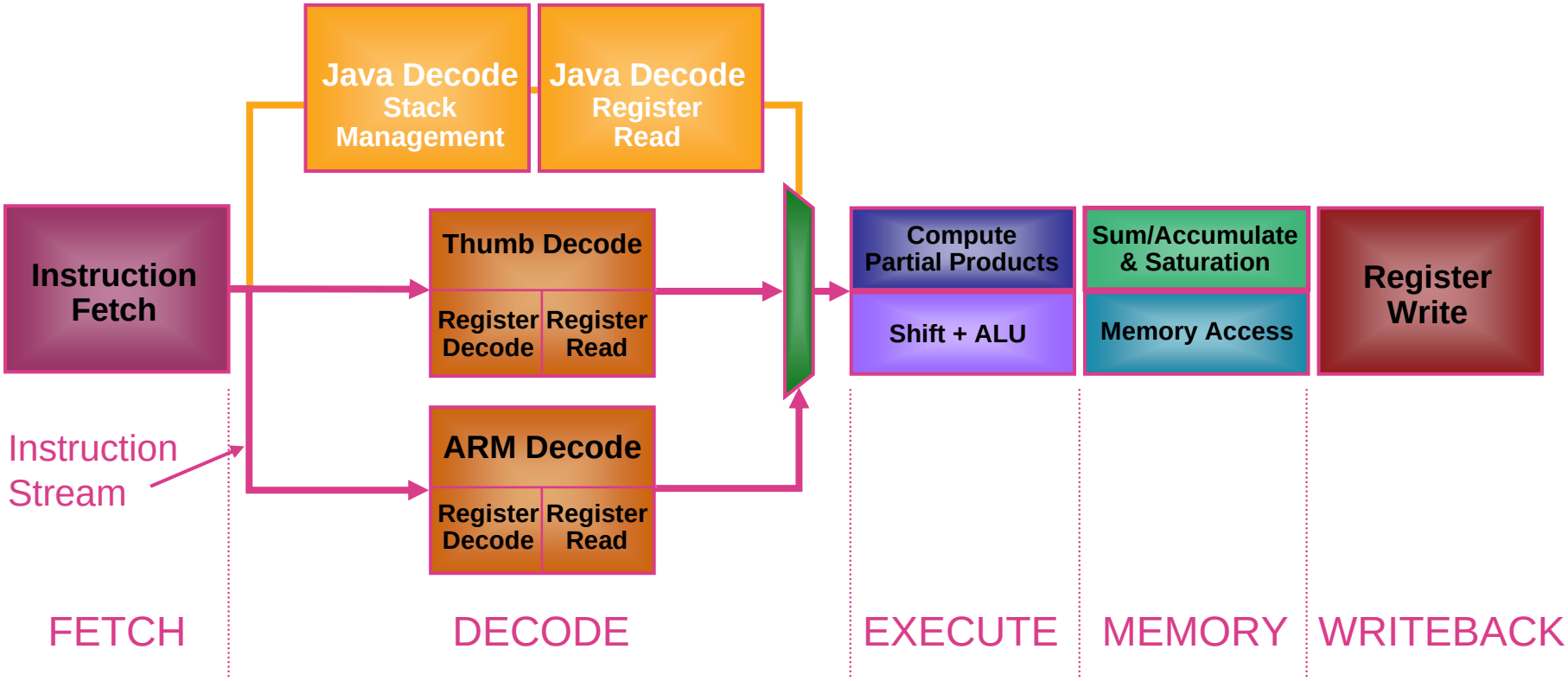
ARMv7TDMI

- Simple 3 stage pipeline
  - Fetch, Decode, Execute
  - Multiple cycles in execute stage for Loads/Stores
- Simple core
  - “Roll your own memory system”



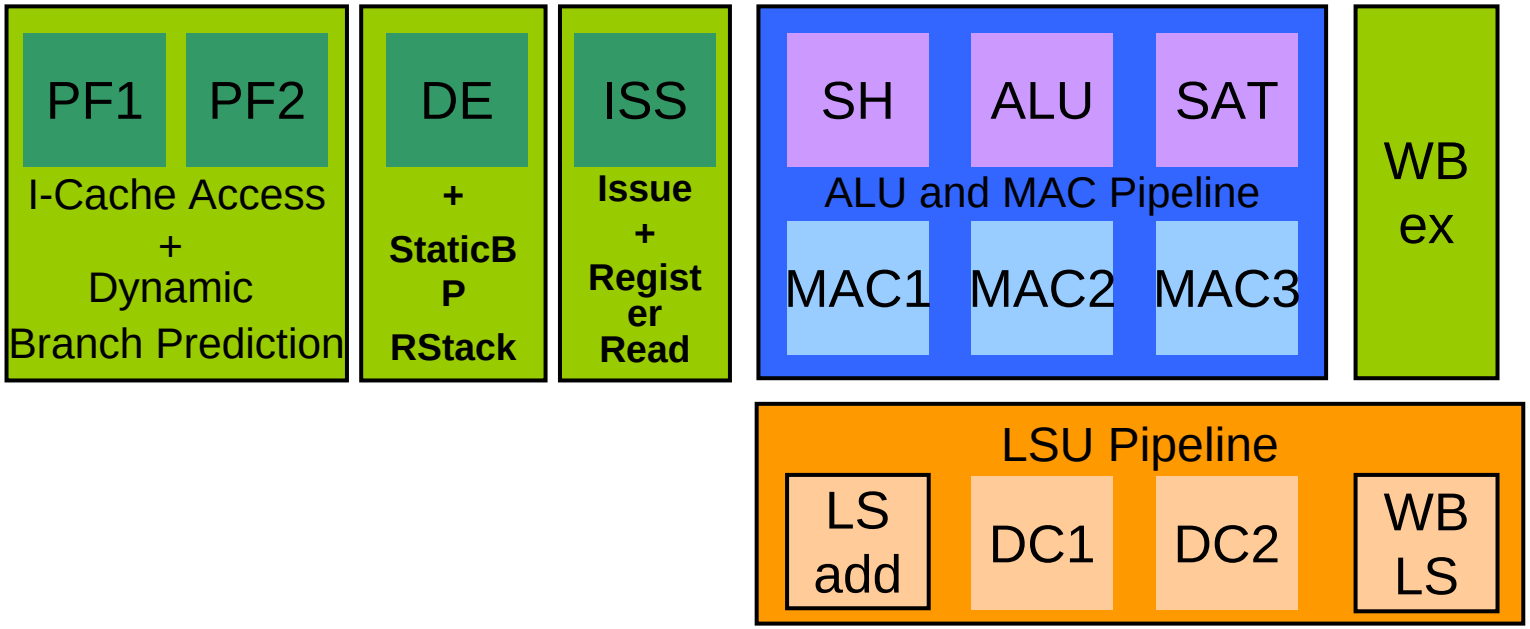
ARM926EJ-S

- 5 stage pipeline single issue core
  - Fetch, Decode, Execute, Memory, Writeback
    - Most common instructions take 1 cycle in each pipeline stage
  - Split Instruction/Data Level1 caches Virtually tagged
  - MMU – hardware page table walk based



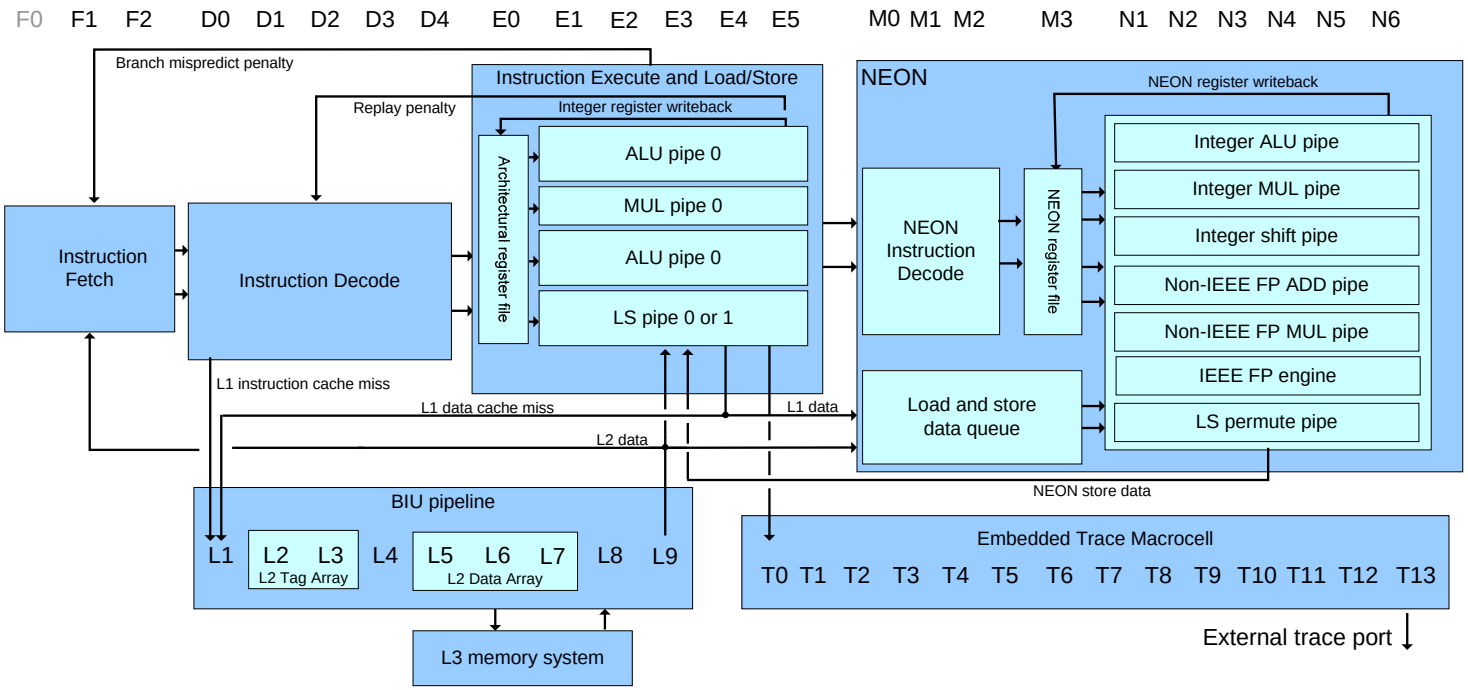
ARM1176JZF-S

- 8 stage pipeline single issue
  - Split Instruction/Data Level1 caches Physically tagged
    - Two cycle memory latency
  - MMU – hardware page table walk based
  - Hardware branch prediction



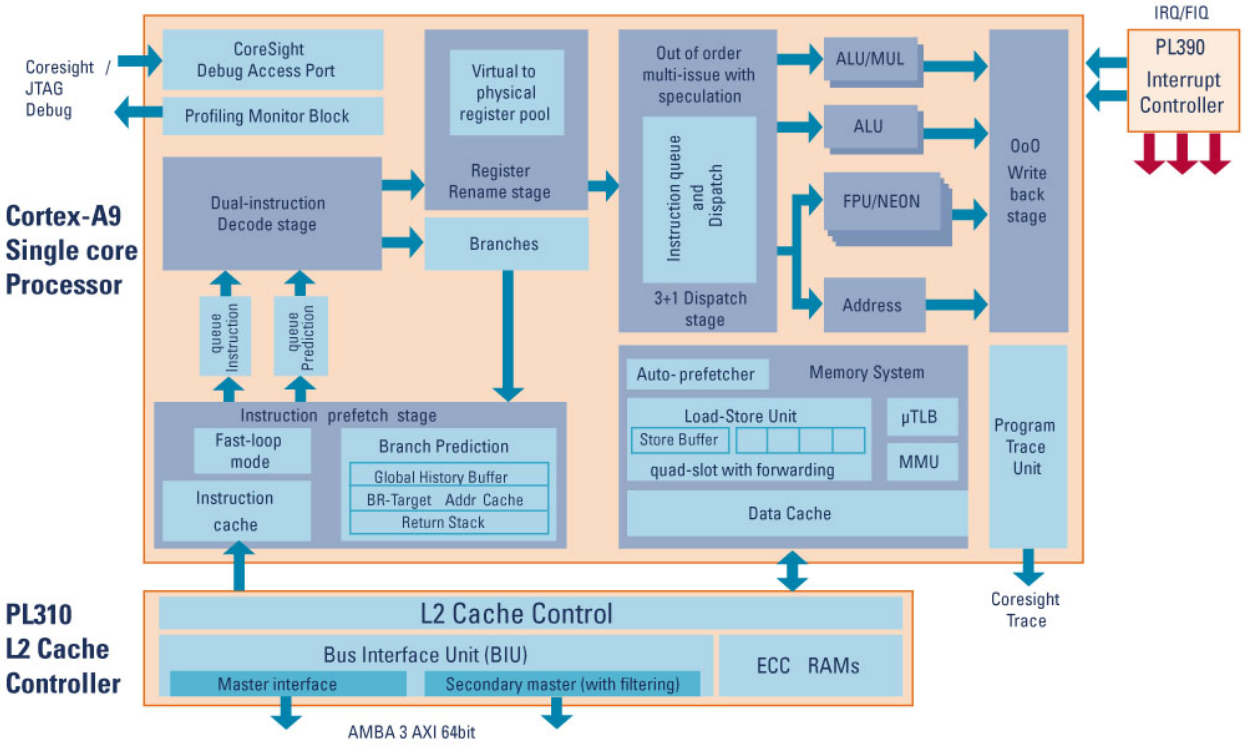
Cortex-A8

- Dual Issue, in-order
  - 10 stage pipeline (+ Neon Engine)
- 2 levels of cache – L1 I/D split, L2 unified
- Aggressive Branch Prediction



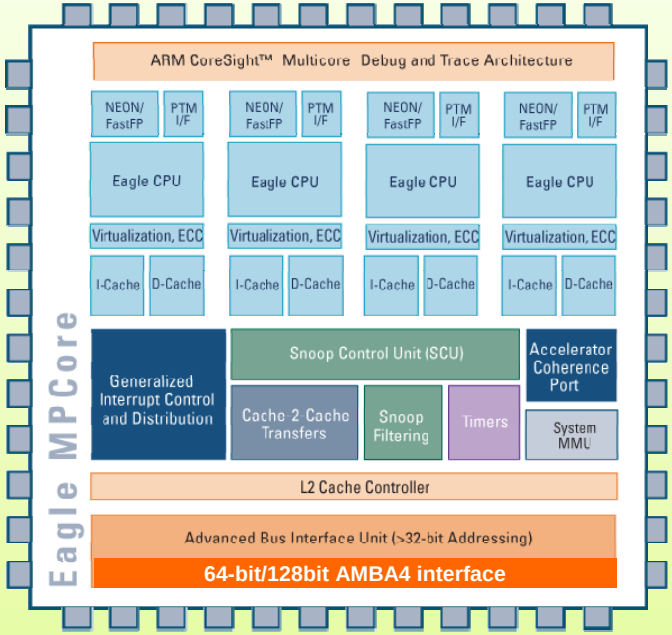
Cortex-A9

- Dual Issue, out-of-order core
- MP capable – delivered as clusters of 1 to 4 CPUs
  - MESI based coherency scheme across L1 data caches
  - Shared L2 cache (PL310)
  - Integrated interrupt controller

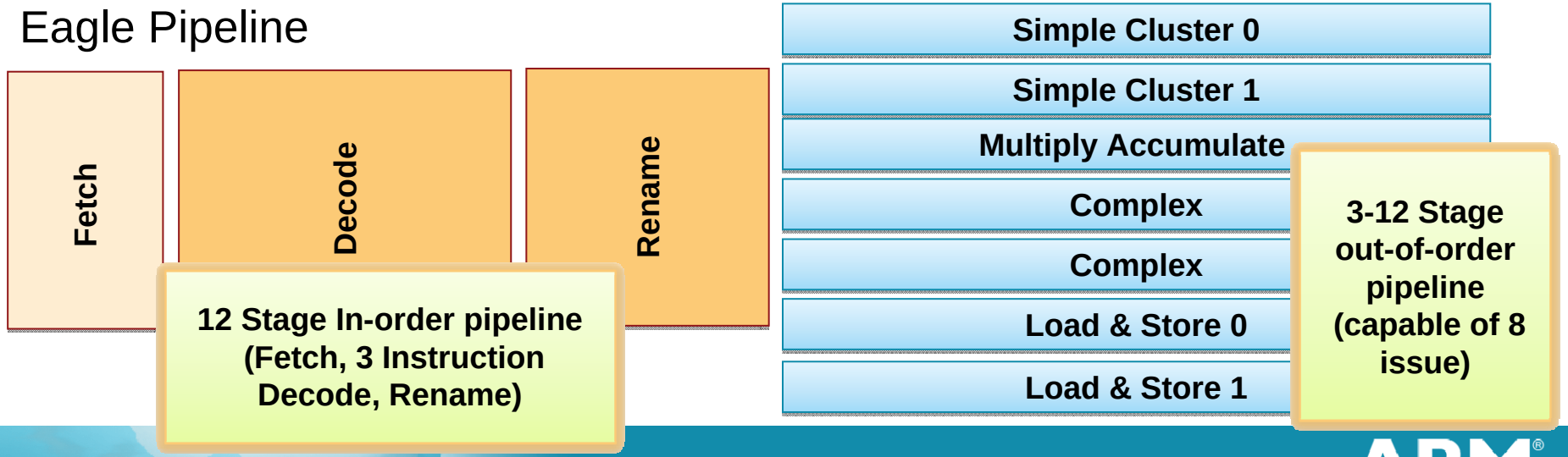


Cortex-A15 – Just Announced - Core Detail

- 2.5Ghz in 28 HP process
  - 12 stage in-order, 3-12 stage OoO pipeline
  - 3.5 DMIPS/Mhz ~ 8750 DMIPS @ 2.5GHz
- ARMv7A with 40-bit PA
- Dynamic repartitioning Virtualization
  - Fast state save and restore
  - Move execution between cores/clusters
- 128-bit AMBA 4 ACE bus
  - Supports system coherency
- ECC on L1 and L2 caches



Eagle Pipeline



Just some basic lessons from experience

- Architecture is part art, part science
  - There is no one right way to solve any particular problem
    - ….Though there are plenty of wrong ways
  - Decision involves balancing many different criteria
    - Some quantitative, some more subjective
  - Weighting those criteria is inherently subjective
- Inevitably architectures have fingerprints of their architects

- Hennessy & Patterson quantitative approach is excellent
  - But it is only a framework for analysis – a great toolkit

- Computer *science* – we experiment using benchmarks/apps
  - *…but the set of benchmarks/killer applications is not constant*

- *Engineering* is all about technical compromise, and balancing factors
  - The art of ***Good Enough***

First Lesson – It's all about Compatibility

- Customers absolutely expect compatibility
  - Customers buy your roadmap, not just your products
  - The software they write is just expected to work
    - Trying to obsolete features is a long-term task

- Issues from the real world:
  - Nobody actually really knows what their code uses
    - …and they've often lost the sources/knowledge
  - People don't use features as the architect intended
    - Bright software engineers come up with clever solutions
    - The clever solutions find inconvenient truths
  - Compatibility is with what the hardware actually does
    - Not with how you wanted it to be used
    - There is a thing called "de facto architecture"

Second lesson – orthogonality

- Classic computer texts tell you orthogonality is good

- Beware false orthogonalities
  - ARM architecture R15 being the program counter
    - Orthogonality says you can do lots of wacky things using the PC
    - On a simple implementation, the apparent orthogonality is cheap
  - ARM architecture has “shifts with all data processing”
    - Orthogonality from original ARM1 pipeline
    - But the behaviour has to be maintained into the future
  - Not all useful control configurations come in powers of 2

- *Fear* the words “It just falls out of the design”
  - True for today’s microarchitecture – but what about the next 20 years?
  - Try to only architect the functionality you think will actually be useful
    - Avoid less useful functionality that emerges from the micro-architecture

Third Lesson – Microarchitecture led features

- Few successful architectures started as architectures
  - Code runs on implementations, not on architectures
  - People buy implementations, not architectures
    - …IP licensing notwithstanding
- Most architectures have “micro-architecture led features”
  - “it just fell out”
  - Optimisations based on first target micro-architecture
    - MIPS – delayed branch slots
    - ARM – the PC offset of 2 instructions
  - Made the first implementation cheaper/easier than the pure approach
    - …but becomes more expensive on subsequent implementations
- Surprisingly difficult trade-off
  - Short-term/long-term balance
    - Meeting a real need sustainably vs overburdening early implementations

Fourth Lesson: New Features

- Successful architectures get pulled by market forces
  - Success in a particular market adds features for that market
    - Different points of success pull successively over time
    - Solutions don't necessarily fit together perfectly
  - Unsuccessful architectures don't get same pressures
    - …which is probably why they appear so clean!

- Be very careful adding new features:
  - Easy to add, difficult to remove
    - Especially for user code
  - "Trap and Emulate" is an illusion of compatibility
    - Performance differential is too great for most applications

Lessons on New Features

- New features rarely stay in the application space you expect….
  - …Or want – architects are depressingly powerless
    - Customers will exploit whatever they find
  - So worry about the general applicability of that feature
    - Assume it has to go everywhere in the roadmap
- If a feature requires a combination of hardware and specific software…
  - …be afraid – development timescales are different
  - Be **very** afraid of language specific features
    - All new languages appear to be different…
      - ….but very rarely are
- Avoid solving this years problem in next year's architecture
  - ... Next year's problem may be very different
  - Point solutions often become warts – all architectures have them
  - If the feature has a shelf-life, plan for obsolence
    - Example: Jazelle-DBX in the ARM architecture

Thoughts on instruction set design

- What is the difference between an instruction and a micro-op?
  - RISC principles said they were the same
    - Very few PURE RISC architectures exist today
    - ARM doesn't pretend to be "hard-core" RISC
      - …I'll claim some RISC credentials
  - Choice of Micro-ops is micro-architecture dependent
    - An architecture should be micro-architecturally independent
    - Therefore mapping of instructions to micro-ops is inherently "risky"
- Splitting instructions easier than fusing instructions
  - If an instruction can plausibly be done in one block, might be right to express it
    - Even if some micro-architectures are forced to split the instruction
  - But, remember legacy lasts for a very long time
    - Imagine explaining the instructions in 5 years time
- Avoid having instructions that provide 2 ways to do much the same thing
  - Everyone will ask you which is better - lots of times…
- If it feels a bit clunky when you first design it…
  - …. it won't improve over time

Final point – Architecture is not enough

- **Not Enough to ensure perfect "write once, run everywhere"**

- Customers expectations of compatibility go beyond architected behaviour
  - People don't write always code to the architecture
    - …and they certainly can't easily test it to the architecture
    - ARM is developing tools to help address this
      - Architecture Envelope Models – a suite of badly behaved legal implementations
  - The architecture defines whether they are software bugs or hardware incompatibilities
    - …allows you to assign blame (and fix the problem consistently)

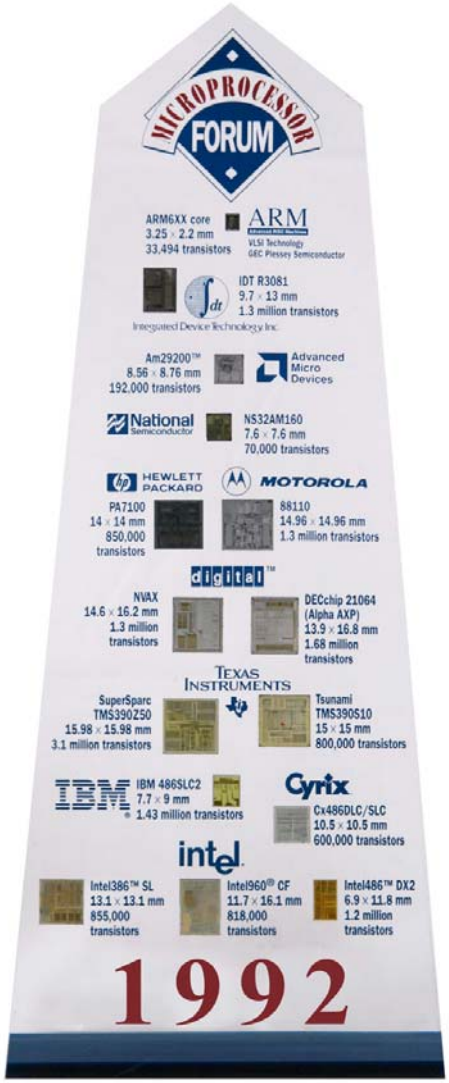
- Beware significant performance anomalies between architectural compliant cores
  - If I buy a faster core, I want it to go reliably faster…without recompiling

- Multi-processing introduce huge scope for functional differences from timing
  - Especially in badly synchronised code
  - Concurrency errors

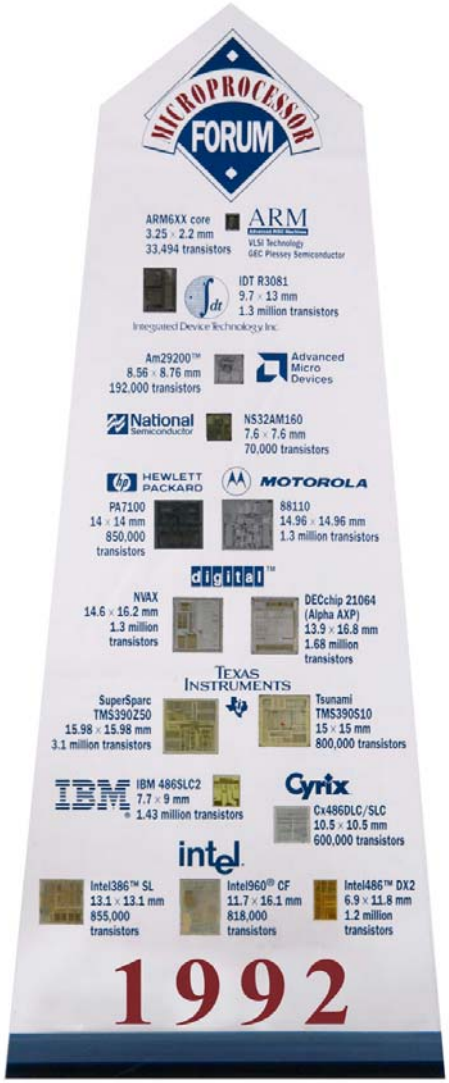
- BUT THE ARCHITECTURE IS A STRATEGIC ASSET

History – the Passage of Time

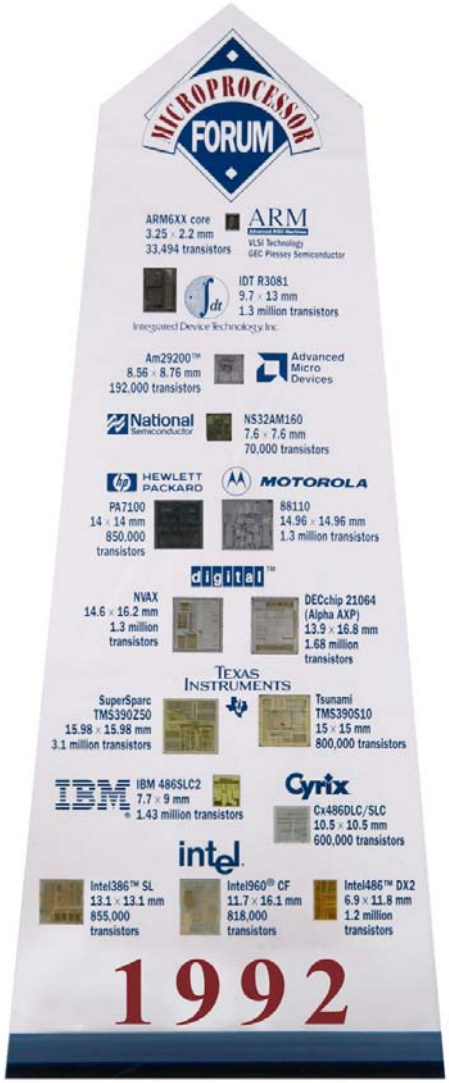
Microprocessor Forum 1992

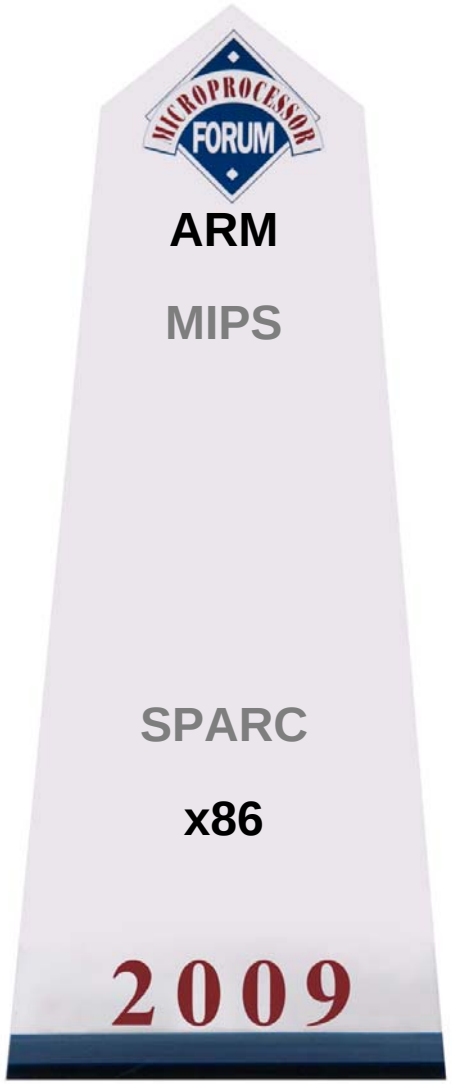


Count the Architectures (11)



The Survivors – At Most 2





What Changed?

- Some business/market effects
- Some simple scale of economy effects
- Some technology effects
- Adoption and **Ecosystem** effects

- It wasn't all technology – not all of the disappearing architectures were bad
  - Not all survivors are good!
  - Not all good science succeeds in the market!

- Never forget "**Good eno**ugh"

Thank you

Questions