arm

Arm Cortex-A9 processor

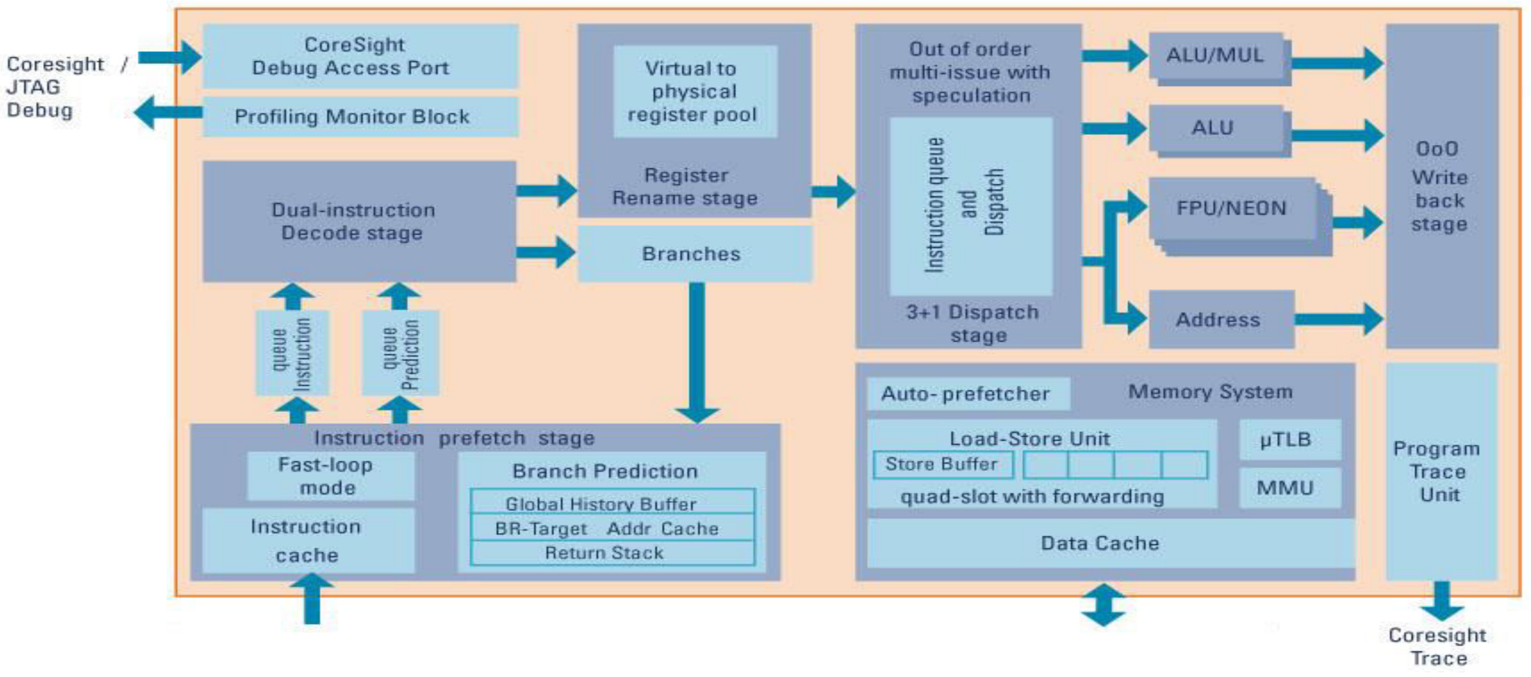
Learning Outcomes

At the end of this module, you will be able to:

- Describe the configuration and features of the Arm Cortex-A9 processor.
- Outline the purpose and properties of Register Renaming, Virtual Flag Registers, Out of Order Issue and Small Loop Mode in the Arm Cortex-A9 processor.
- Identify the properties of Program Flow Prediction in the Arm Cortex-A9 processor.
- Explain the function and operation of a Performance Monitoring Unit and Memory Management Unit.
- Outline the memory and cache properties in the Arm Cortex-A9 processor.

Arm Cortex-A9

- Armv7-A architecture
  - Thumb-2, ThumbEE
- Synthesizable
- High-performance core:
  - Variable-length multi-issue pipeline
  - Register renaming
  - Speculative data prefetching
  - Branch prediction and return stack
- 64-bit AXI instruction and data inter
- TrustZone extensions
- L1 data and instruction caches
  - 16–64KB each
  - Four-way set-associative

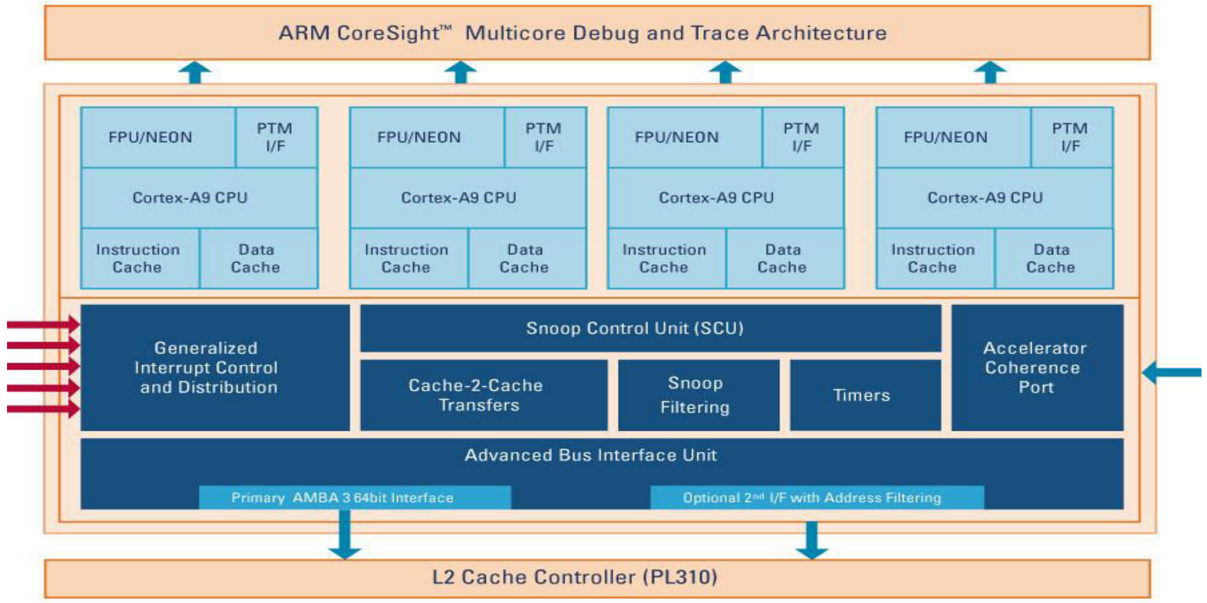


Optional features

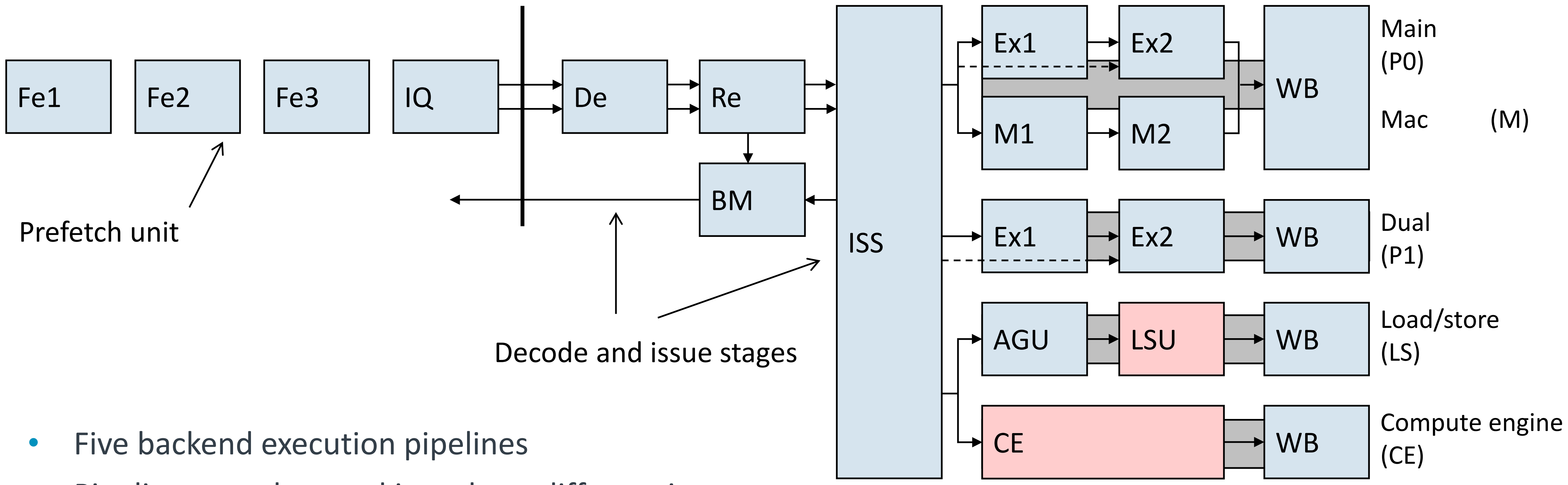
- PTM Program Flow Trace interface
- IEM power-saving support
- Full Jazelle DBX support
- VFPv3-D16 floating-point unit (FPU) or Neon™ media processing engine

Cortex-A9 MPCore

- Contains up to 4 Cortex-A9 processors
  - Identical to standard single-core processors described in this module
  - Private timer and watchdog unit per processor
- Snoop control unit (SCU)
  - Maintains L1 data cache coherency between processors
  - Arbitrates accesses to the L2 memory system, through one or two external 64-bit AXI Managers interfaces
  - Optional Accelerator Coherency Port for maintaining coherency with DMA controller, graphics processor, or similar
- Integrated interrupt controller
  - Same programmer's model as Arm's Generic Interrupt Controller (GIC): the PL390 PrimeCell


ARM CoreSight™ Multicore Debug and Trace Architecture
FPU/NEON
PTM I/F
Cortex-A9 CPU
Instruction Cache
Data Cache
Generalized Interrupt Control and Distribution
Snoop Control Unit (SCU)
Cache-2-Cache Transfers
Snoop Filtering
Timers
Accelerator Coherence Port
Advanced Bus Interface Unit
Primary AMBA 3 64bit Interface
Optional 2nd I/F with Address Filtering
L2 Cache Controller (PL310)

Cortex-A9 Pipeline



- Five backend execution pipelines
- Pipelines are clustered into three different issue groups.
  - Main or multiply accumulate (Mac)
  - Dual execution (also known as secondary)
  - Load/store, or compute engine (Neon or floating point)

Core can issue up to three instructions per cycle.

Cortex-A9 MPE Configuration

- Cortex-A9 can be configured as follows:
  - Without MPE (no VFP or Neon support)
  - VFP/FPU only (sixteen 64-bit double-precision registers)
  - VFPv3 and Neon (thirty-two 64-bit double-precision registers)
- RVCT support
- Arm DS-5 and DSTREAM support

Cortex-A9 Media Processing Engine

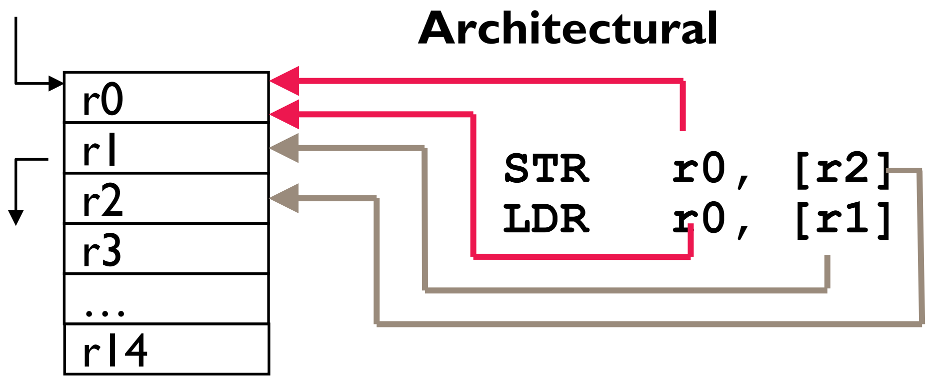
- Cortex-A9 Neon Media Processing Engine (MPE)
  - Support for Advanced SIMD and VFPv3
  - Pipelined (shares backend execution stage with load/store instructions)
  - Large register file (can be implemented differently for VFP only)
  - Thirty-two 32-bit S (single) registers
  - Thirty-two 64-bit D (double) registers
  - Sixteen 128-bit Q (quad) registers
  - Neon and floating-point unit can be enabled/disabled in software for power savings.
- VFPv3
  - No floating point support code required for most operations
  - Vector (floating point) operations no longer supported in hardware
  - Register bank optimized to sixteen double-precision registers for size and power considerations and compatibility with VFPv2/Arm11 software
  - Emphasis on fast single-precision support
  - Full compliance with IEEE 754 standard

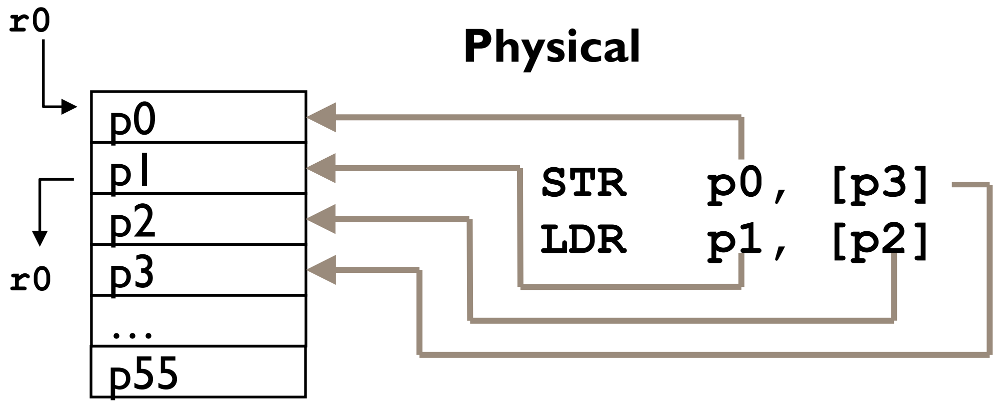
Register Renaming

- Cortex-A9 has two classes of core registers: Architectural and physical
  - Architectural registers (r0–r15, CPSR, etc.) that are visible to software
  - Physical registers: physically implemented in the CPU but not visible to software
  - There are also fifty-six general-purpose physical registers, and eight flag registers (for CPSR)

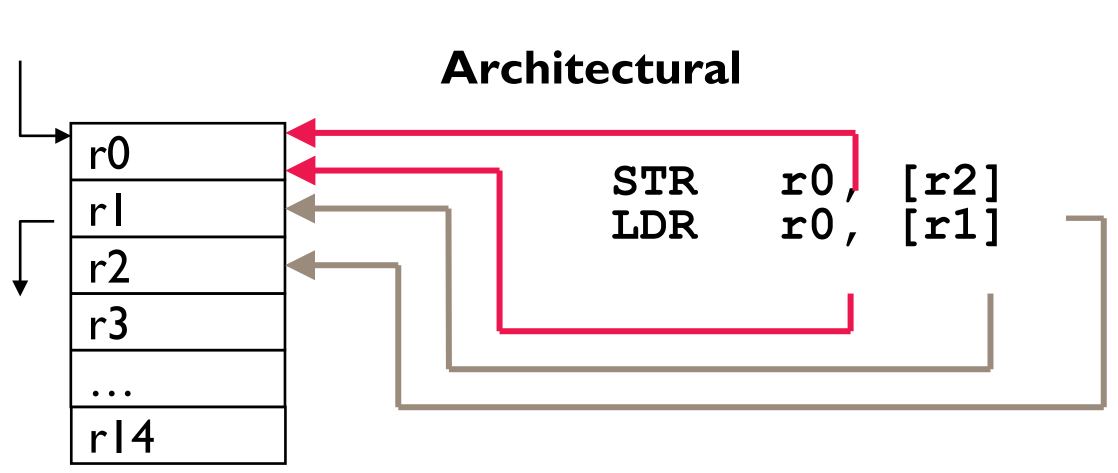
Architectural registers are allocated from a pool of physical registers in the execute stages of the pipeline.

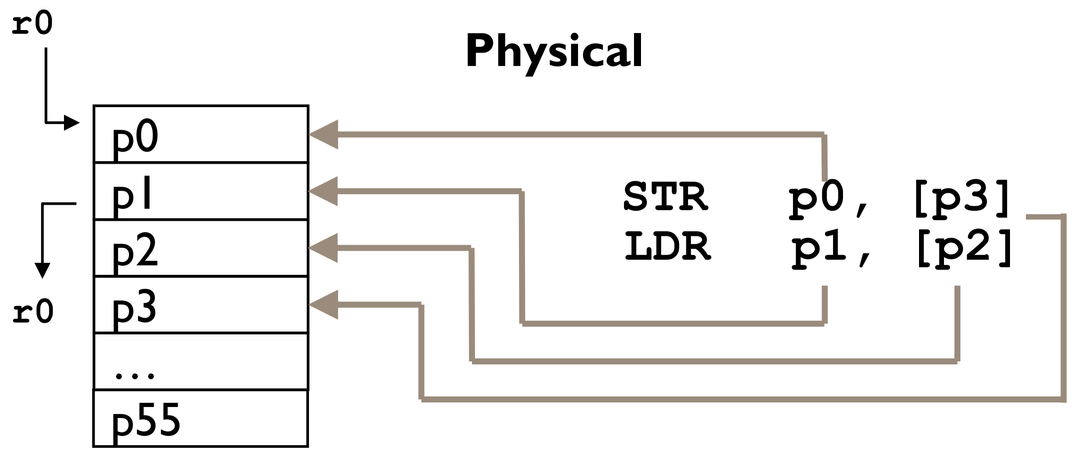
  - An architectural register can be mapped to multiple physical registers.
  - Removes interlocks due to register dependencies
  - Can still interlock due to a data dependency





Register Renaming Example

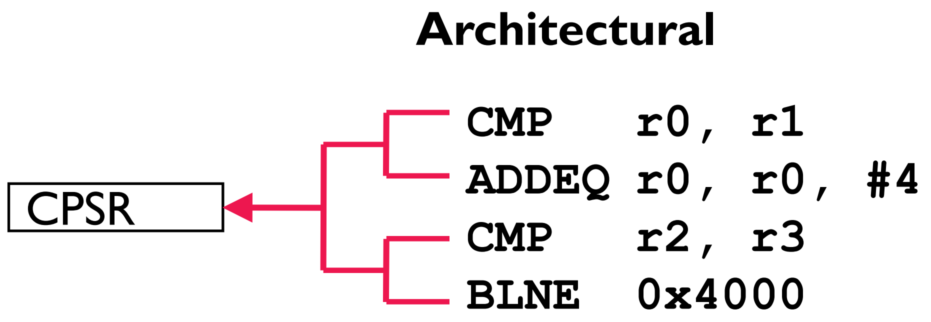




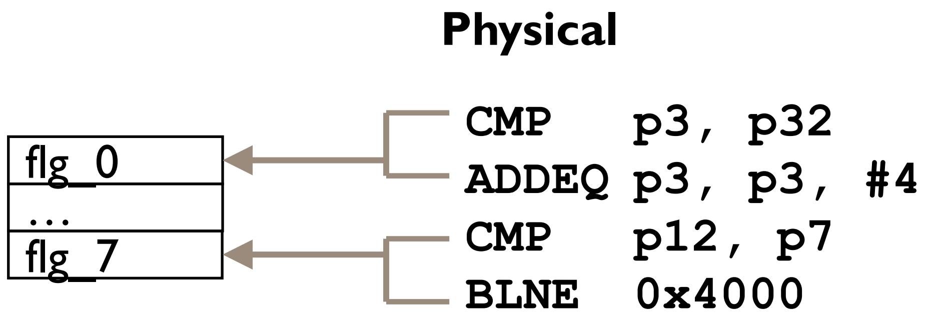
- The architectural view shows the situation as on a core without renaming.
  - Once a register value has been written to memory, that register is available to the core.
  - In this example, r0 would not be accessible in the register bank until the STR had completed: even though there is no dependency.
- Register renaming removes this register dependency.
  - As the two versions of r0 are fundamentally not connected, they are assigned to two different physical registers.
  - From the processor's point of view, the dependency no longer exists.

Virtual Flags Registers

- The virtual flags registers perform limited renaming of the CPSR.
  - They hold copies of the flag bits of CPSR: NCZV, GE, Q.
- Can remove register dependencies on CPSR flags
  - Allows dual-issue in certain cases
- The flags registers do not (cannot) hold copies of state/mode bits.
- Example: removing a stall caused by a CMP following a conditionally executed instruction

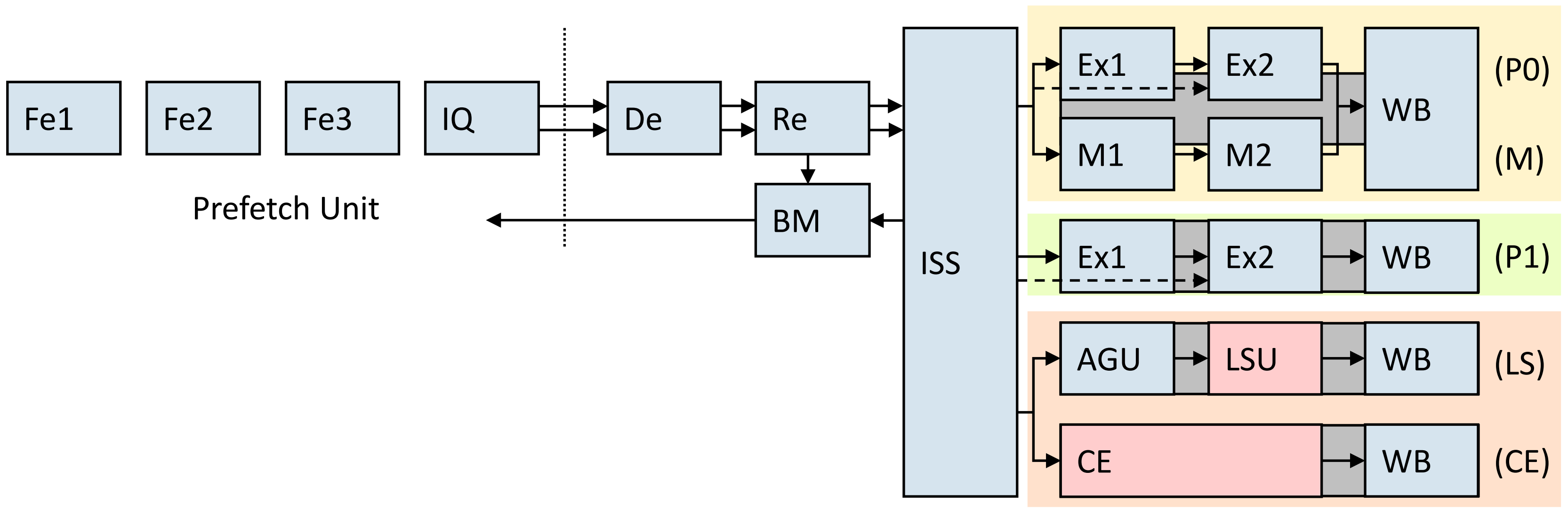


```
CMP   r0, r1
ADDEQ r0, r0, #4
CMP   r2, r3
BLNE  0x4000
```



```
CMP   p3, p32
ADDEQ p3, p3, #4
CMP   p12, p7
BLNE  0x4000
```

Out of Order Issue/Completion

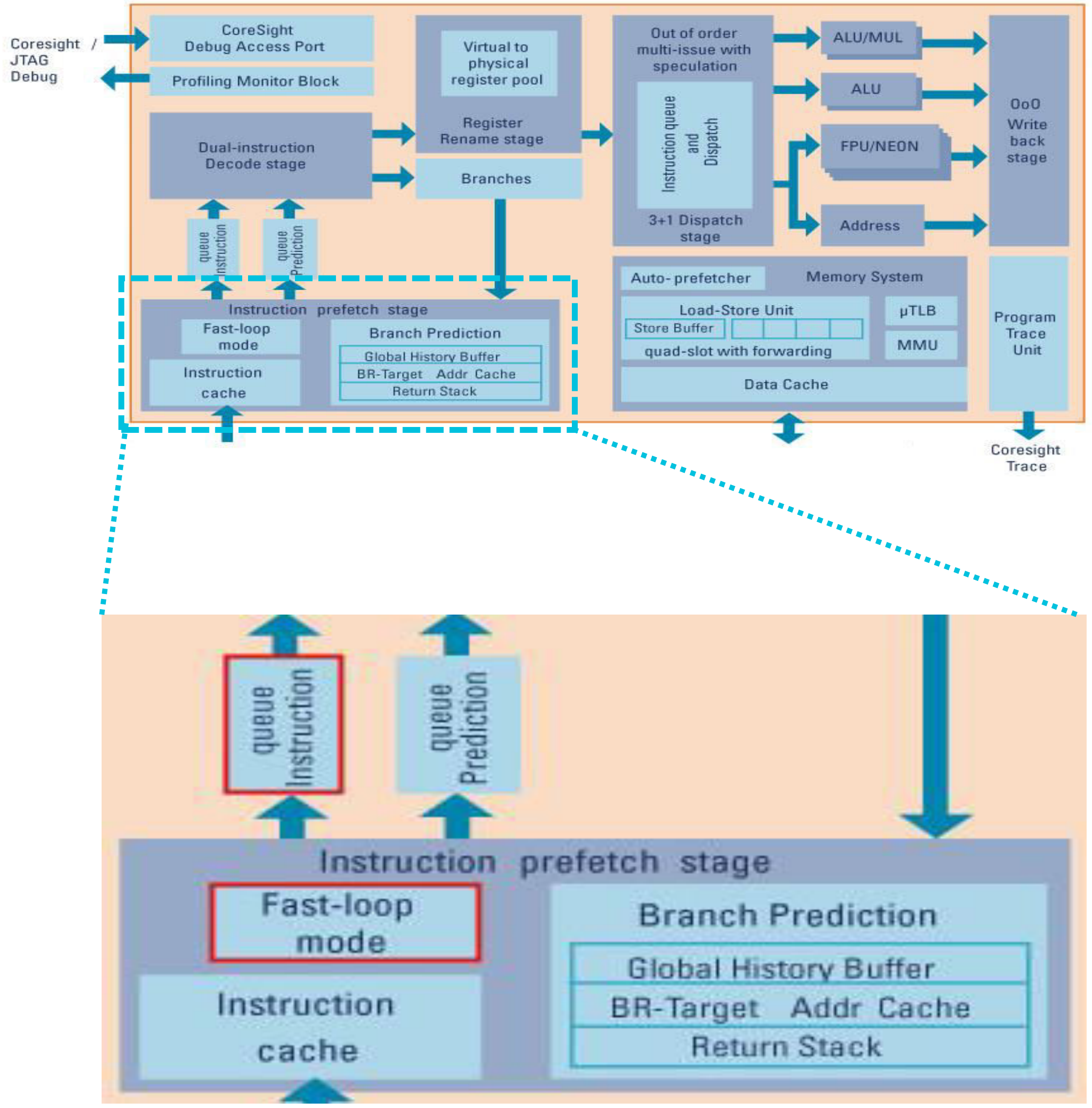


- The core can decode up to two instructions per cycle, but can issue up to three instructions per cycle.
  - Due to buffering in issue stage
  - With two already renamed instructions additionally buffered in rename stage

Register renaming allows instructions in different pipelines to complete out-of-order where no data dependencies exist.

Small Loop Mode

- Allows a small loop to execute entirely out of the IQ
  - The prefetch unit stops fetching instructions from th cache.
  - Lowers power consumption
- Small loop mode can be activated if
  - Loop fits inside two cache lines.
  - Code is all Arm, or all Thumb-2.
  - Loop ends with a conditional backwards branch.



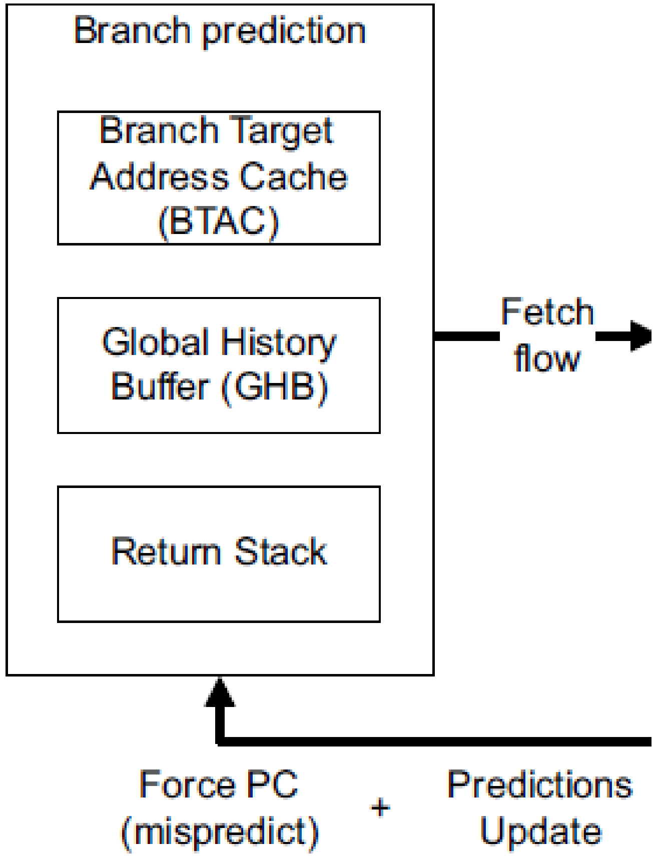
Program Flow Prediction

- Dynamic branch prediction
  - Branches assigned one of four states: strongly/weakly, taken/not taken
  - Global history buffer maintains state of the last 4096 branches*.

Branch target address cache (BTAC): 512-entry* cache of taken branch addresses

- BTAC must be flushed on reset or context switch.
- More than two likely-taken instructions in 16-byte blocks will harm performance (because of BTAC structure).

* Numbers quoted in asterisk may vary depending on the implementation of the Cortex-A9 processor.



Program Flow Prediction

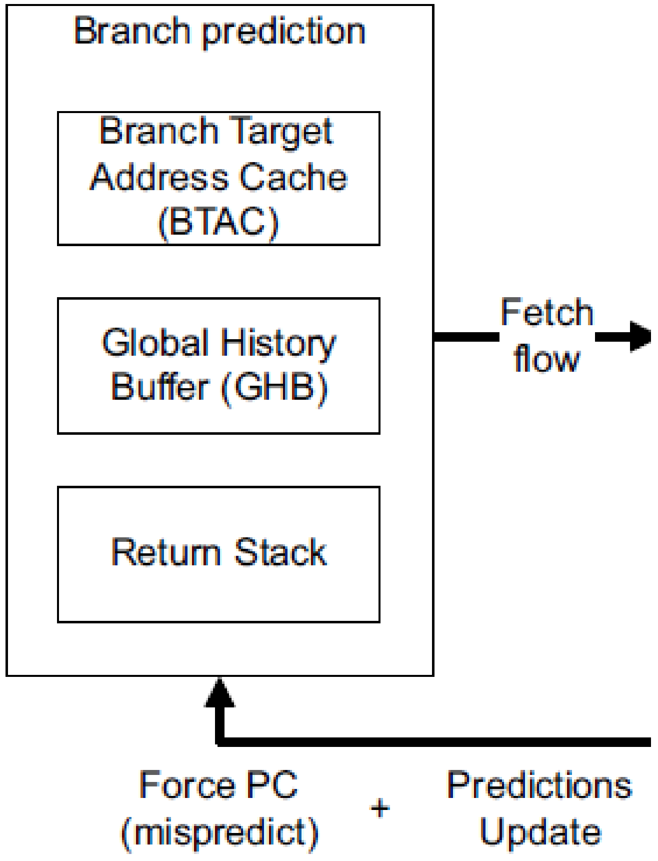
Eight-entry return stack

- Predicts returns from subroutines based on fetching certain return instructions

Branch prediction is enabled by cp15 System Control Register Z bit

- Disabled at reset: must be explicitly enabled by programmer

Performance monitors can check efficiency of branch prediction.



Performance Monitoring Unit (PMU)

- The Performance Monitoring Unit provides a non-intrusive method of collecting execution information from the core.
  - Enabling the PMU does not affect the performance or program behavior.

The PMU provides

- A dedicated cycle counter: counts execution cycles (optional 1/64 divider)
- Programmable event counters
- Counters can record items like cache/TLB misses, branch prediction performance, pipeline stalls, and memory accesses.
- The Cortex-A9 PMU provides six configurable event counters.

The PMU can be configured to generate interrupts if a counter overflows.

- Interrupt signals are an output from the core, into an external interrupt controller.

Cortex-A9 Supports Armv7-A Architecture

- Mixed-endian support for data side accesses
  - Little-endian (LE) or big-endian (BE-8/byte-invariant)
  - Data access endianness is controlled by the E bit in the CPSR.
  - Endianness on exception entry is controlled by the EE bit in the CP15 c1 control register.

Unaligned access support in hardware (for normal memory)

  - Accesses to unaligned addresses are handled in hardware.
  - Controlled by the A bit in the CP15 c1 control register.
  - NEON unaligned access based on memory type (normal, device, or strongly ordered)

Cache and MMU support

  - Includes Translation Lookaside Buffers (TLBs)

Level 1 Memory System

Caches

- 16-64KB, 4-way set-associative

Data side

- Dedicated PLD unit for handling data preload requests
- Speculative data prefetching
- Merging store buffer

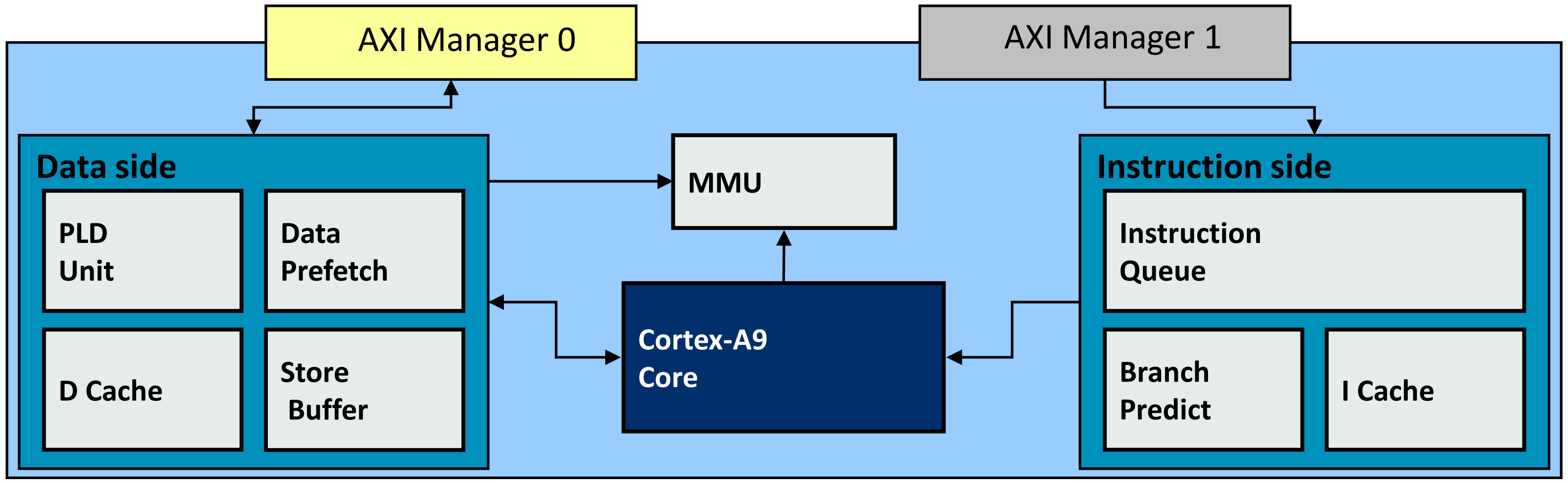
Optional support for parity checking

Instruction side

- Branch prediction and instruction queue

MMU

- Separate I- and D- MicroTLBs
- Unified main TLB
- Support for translation table walks through L1 D-cache



Load/Store Features

- Load/store instructions can be issued speculatively.
  - Before condition of instruction, or preceding branch, has been resolved, or before data to be written has become available
  - Any side effects, such as modified registers, are flushed if condition fails.

PLD support implemented in separate unit

  - Can handle up to four outstanding PLD operations (explicit loads/stores have higher priority)

Speculative data prefetching

  - Monitors sequential accesses made by program and starts fetching next expected line before it has been requested
  - Enabled in cp15 auxiliary control register (DP bit)
  - Prefetched lines can be dropped before allocation, and PLD has higher priority.

Store buffer is a merging store buffer with four 64-bit slots.

  - Data forwarding: data can be moved directly from LDR to STR in the LSU.

Caches

- Four-way set-associative
  - 16, 32, or 64 KB each for data and instructions independently (synthesis option)
  - Eight words per line
  - Data cache is physically indexed and physically tagged.
  - Instruction cache is virtually indexed and physically tagged.

No lockdown support and no TCM

  - Instead, use L2 cache lockdown or fast on-chip memory.

Support for parity checking (synthesis option)

Round robin or random replacement policy

  - Victim counter is read at the time of miss, not time of allocation, and then incremented at the time of allocation.
  - An invalid line in the set will be replaced in preference to using the victim counter.

Caches need to be invalidated before they are enabled (after reset).

  - Different from previous Arm cores

Data Cache

- Non-blocking
  - Load/store can continue to hit in cache while it is performing allocations from external memory due to prior read/write misses.
  - Supports four outstanding read misses and four outstanding write misses.

Contains the local load/store exclusive monitor

  - Used for LDREX/STREX synchronization
  - Monitors one address only, with eight-word (one cache line) granularity
  - Avoid interleaving LDREX/STREX sequences.
  - Always execute a CLREX instruction as part of any context switch.
  - Note: Non-exclusive store to tagged location does not clear monitor state.

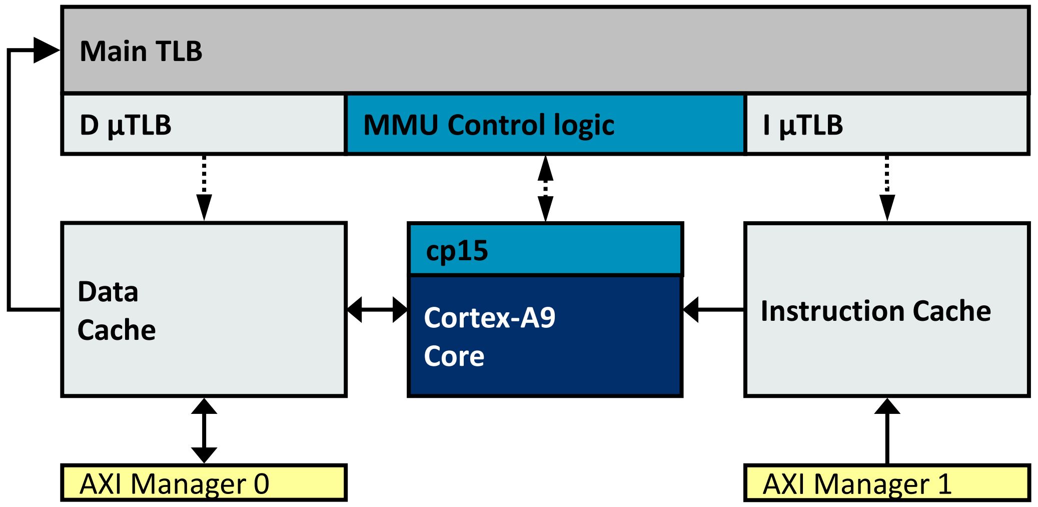
Data Cache

Support for exclusive cache operation (disabled by default)

- A cache line is valid only in L1 **or** L2 cache, never in both at the same time.
- Line fill into L1 causes line to be marked invalid in L2.
- Eviction from L1 causes allocation in L2 (even if not dirty).
- Line fill into L1 from dirty L2 line forces eviction to external memory.
- Increases cache utilization, reducing power consumption.

Memory Management Unit

- Translation table walks can be configured to go through L1 data cache.
  - Allows page tables to be cached
  - Configured in TTBRx IRGN bits, so can be individually configured per process (TTBR0) and also for system region (TTBR1)



Main TLB should be invalidated before MMU is enabled after reset.

- MicroTLBs automatically invalidated by any TLB operation, or change of ASID

Armv7 Architecture Effects

- Old CP15 c7 wait for interrupt operation has been removed.
  - Executes as a NOP
  - Use WFI instruction instead.

L1 caches no longer guaranteed to be invalidated on reset

  - Must be manually invalidated

L1 cache lockdown is optional.

  - Not implemented in Cortex-A9