

ARM Cortex-A* Series Processors

Haoyang Lu, Zheng Lu, Yong Li, James Cortese

1. Introduction

With low power consumption, the ARM architecture got popular and 37 billion ARM processors have been produced as of 2013, up from 10 billion in 2008. The ARM architecture (32-bit) is the most widely used architecture in mobile devices, and most popular 32-bit one in embedded systems. The recently introduced ARMv8-A architecture add support for a 64-bit address space and 64-bit arithmetic. Apple first implemented the ARMv8-A architecture in the Apple A7 chip in the iPhone 5S in 2013. The ARM Cortex-A series processors are one branch of the ARM family which are designed for running operating systems, predominantly on mobile internet devices. They are used in a large variety of smart phones, notebooks, tablets, eBook readers, smart television sets, and digital set top boxes for digital televisions. They are also used in some network controllers, printers, and servers. The various processors of the Cortex-A* series and their applications are as follows (Table 1)

Table 1. Applications of Cortex-A* series processors

	A5	A7	A8	A9	A15	A53	A57
Smart phones	*		*	*		*	*
Home Computing	*		*	*			
Smart TVs	*		*	*			
Digital Cameras	*			*			
Embedded Computing	*	*	*	*	*	*	*
Home Networking	*			*	*		
Storage	*						

Generally, these processors can all perform the same tasks. This is because they are general purpose CPUs with a focus on running an operating system. Specific operating systems include various Android flavors, Chrome OS, Embedded Windows, and Linux distributions such as Ubuntu and Debian. They are used in the Raspberry Pi and also power the Ford Sync system found in Ford vehicles. The difference between one Cortex-A* series processor and the next is one of efficiency and

power. Processing power tends to increase as the number increases such as A5 being least powerful and A57 being most powerful. A53 and A57 implement a slightly different architecture than the other A series. The A50s implement ARMv8-A, a 64 bit architecture that has a 64 bit address space and can perform 64 bit arithmetic. Other A series implement ARMv7-A, a 32 bit architecture. The A50s are backward compatible with ARMv7-A instructions.

The market for ARM Cortex A series processors is enormous. Consider the breadth of applications described above to understand why. Nearly all of the processors used in these applications are ARM processors. The ARM Cortex-A series is the most modern series of ARM processors so all cutting edge devices will use A series processors. At least 3 ARM chips are in a typical smartphone. Also, because ARM does licensing as part of its business model, there are many processors that are designed independently of ARM but use the ARM instruction set. These include Snapdragon, Apple A6, Apple A6X, Apple A7, X-Gene, and Denver cpus. The Apple A6, A6X, and A7 power Apple iPods, iPads, and iPhones, including the new iPhone 5S.

2. Instruction Set of ARM Cortex A Series

1) ARM and Thumb Instruction sets

The ARM processor has 2 instruction sets, the traditional ARM set, where the instructions are all 32-bit long, and the more condensed Thumb set, where most common instructions are 16-bit long (and some are 32-bit long, introduced by Thumb-2). Only one set can be active at one time. Once the processor is switched into Thumb mode, all instructions will be decoded as using Thumb instead of ARM. Although their instructions are different, they share similar functionalities. Almost every instruction in Thumb encoding has a corresponding encoding in ARM.

- Thumb: To compress the code-density, Thumb technology was introduced in the ARM7TDMI. Its featured Thumb instruction set is a compact 16-bit encoding for a subset of the ARM instruction set. In thumb, the 16-bit opcodes have less functionality. For example, opcodes are restricted to accessing only half of the general-purpose registers. The shorter opcodes give improved code density overall, even though some operations require extra instructions.
- Thumb-2: Thumb-2 extends the limited 16-bit instruction set of Thumb with additional 32-bit instructions. Bit-field manipulation, table branches and conditional execution can be executed under Thumb-2 technology. A novel Unified Assembly Language (UAL) supports generation of either ARM and Thumb instructions from the same source code.

- Thumb Execution Environment (ThumbEE): First appearing in the Cortex-A8 processor, ThumbEE makes small changes to the Thumb-2 to make the instruction set particularly suited to code generated at runtime in managed Execution Environments. ThumbEE is a target for languages such as Java, C#, Perl, and Python, and allows JIT compilers to output smaller compiled code without impacting performance. On 23 November 2011, ARM Holdings deprecated any use of the ThumbEE instruction set, and ARMv8 removes support for ThumbEE.

2) Pipeline of ARM Cortex-A8

The A8 is a dual-issue, statically scheduled superscalar with dynamic issue detection, which allows the processor to issue two instructions per clock. The A8 uses a dynamic branch predictor with a 512-entry two-way set associative branch target buffer and a 4K-entry global history buffer, which is indexed by the branch history and the current PC. A prediction is obtained from the global history buffer when the branch target buffer misses. An incorrect branch prediction results in a 13-cycle penalty. The 13 stage pipeline is shown in Fig 1.

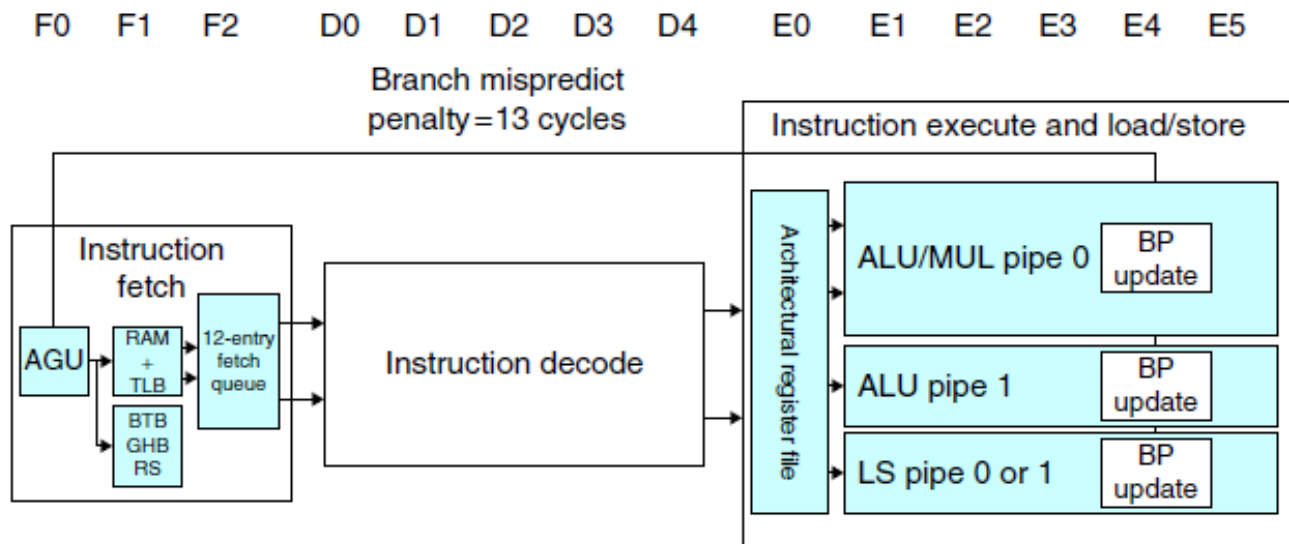


Fig 1. The basic structure of the A8 13-stage pipeline (Note that the F0 Fetch stage is not counted as an official stage in the 13 stage main integer pipeline. This is because ARM processor pipelines have always counted stages beginning with the Instruction Cache access as the first stage.)

Up to two instructions per clock can be issued using an in-order issue mechanism(D1 stage). In D3, a simple scoreboard structure is used to track when an instruction can issue. The 5-stage instruction decode process is shown in Fig 2.

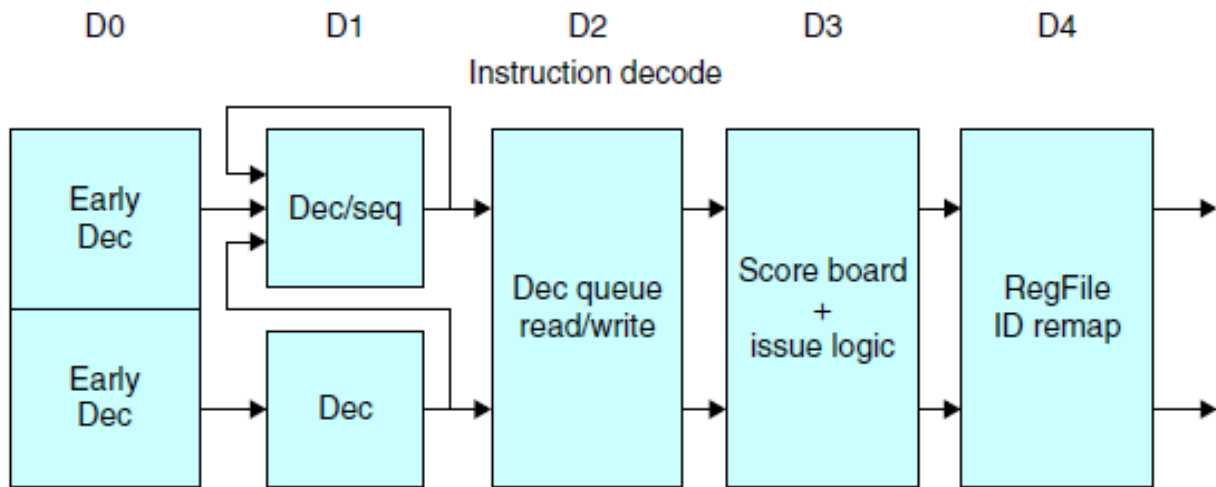


Fig 2. The five-stage instruction decode.

In instruction execute process, either instruction 1 or instruction 2 can go to the load/store pipeline. Fully bypassing is supported among the pipelines.

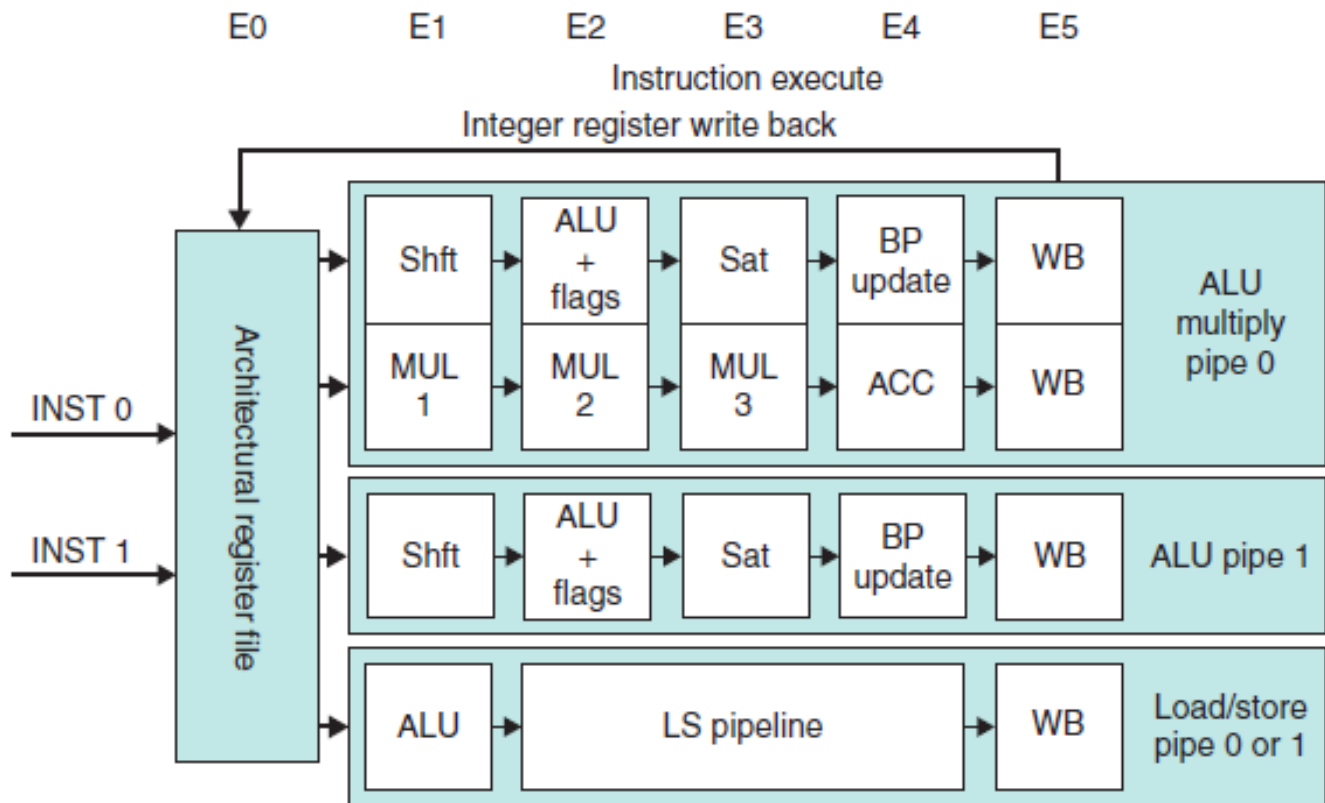


Fig 3. The five-stage instruction decode execution

Due to its dual-issue structure, the ideal CPI of A8 is 0.5. Pipeline stalls can arise from three sources:

1. Functional hazards, which occur when two instructions selected for issue simultaneously use the same functional pipeline.
2. Data hazards, which are detected early in the pipeline and may stall either both instructions
3. Control hazards, which arise only when branches are mispredicted, the penalty is 13 cycle.

ARM Cortex-A9 operates dynamically scheduled superscalar leading out-of-order execution. Up to four pending instructions (two ALUs, one load/store or FP/multimedia, and one branch) can begin execution in a clock cycle. The A9 uses a more powerful branch predictor, instruction cache prefetch, and a nonblocking L1 data cache.

3. Multicore

The Cortex-A5, Cortex-A7, Cortex-A9, Cortex-A12, and Cortex-A15 all support multicore implementations. These processors have ARM MPCore technology that allows for implementations with one to four cores. These multicore implementations are level-1 cache coherent and can be made entirely coherent by using an Accelerator Coherence Port (ACP). This basically allows peripherals and accelerators that do not see the memory as a cached system to use processor caches in a fully coherent manner. These multicore CPUs use a Global Interrupt Controller (GIC) interrupt and communication system which allows up to 224 independent interrupt sources, a large number of connected devices. A Snoop Control Unit (SCU) manages interactions and communications and maintains cache coherence between cores. Operating systems that have a Symmetric Multi-Processing (SMP) mode can fully utilize it using Cortex A series multicore processors.

Similar to multicore is ARM big.LITTLE which is a dual processor setup. In systems that use ARM big.LITTLE, a powerful processor is paired with a less powerful processor. Processing is allocated to either the powerful (big) processor or the little (less powerful) processor depending on the job. This can provide great efficiency benefits (up to 70% in some cases). Available processor pairs for big.LITTLE configurations are the A15 and A7, or the A53 and A57. Each processor involved can be multicore.

4. ARM Exclusive Features of Cortex-A8

1) Advanced SIMD Architecture - NEON coprocessor

The ARM® NEON™ general-purpose SIMD engine efficiently processes current and future multimedia formats, enhancing the user experience [1].

NEON technology is used in ARM Cortex™-A series processors to enhance many multimedia user

experiences, such as watching, editing and enhance videos, game processing, photo processing, and voice recognition. These multimedia applications are among the most common requirements of today's embedded systems such as smartphones and tablets. Advanced SIMD architecture, namely NEON technology in ARM Cortex™-A series processors, adds a set of instructions specialized for several kinds of multimedia processing. It can highly enhance the multimedia and signal processing algorithms which are frequently required by multimedia applications such as video encode/decode, 2D/3D graphics, gaming, audio and speech processing, image processing, telephony, and sound synthesis.

NEON technology is a 128-bit SIMD (Single Instruction, Multiple Data) architecture extension for the ARM Cortex™-A series processors, designed to provide flexible and powerful acceleration for consumer multimedia applications, delivering a significantly enhanced user experience. It has 32 registers, 64-bits wide (dual view as 16 registers, 128-bits wide).

The Advanced SIMD instructions perform packed SIMD operations. The “packed” here means two aspect:

1. Registers are considered as vectors of elements of the same data type. Supported data type including all the common data type in multimedia processing, such as signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single precision floating point [1]. Typically length of registers are 64-bit or 128-bit.
2. Instructions perform the same operation in all lanes. This approach meets the requirement of multimedia applications which need to perform same operations on a large set of data.

The NEON coprocessor can receive up to two valid Advanced SIMD instructions per cycle from the ARM integer instruction execute unit. NEON load data can be retrieved from either the L1 data cache or the L2 memory system. In addition, it can receive 32-bit MCR data from or send 32-bit MRC data to the ARM integer instruction execute unit [2]. NEON is included in all Cortex-A8 devices but is optional in Cortex-A9 devices [4].

2) VFPv3 architecture - VFP coprocessor

ARM Floating Point architecture (VFP) provides hardware support for floating point operations in half-, single- and double-precision floating point arithmetic. It is fully IEEE 754 compliant with full software library support [1].

The hardware floating point architecture provides an enhanced performance of floating point operations used in several popular applications such as automotive powertrain and body control. It is also great for some multimedia applications such as scaling, transforms and font generation in printing, 3D transforms, FFT and filtering in graphics. It can even benefit some emerging applications which are

not widely used these days but are expected to become popular in the following years. Hardware floating point architecture provided by ARM Cortex™-A series processors provides dynamic range and increased precision of floating point operations, which are essential performance indicator in mission-critical applications.

The VFP coprocessor in ARM Cortex™-A series processors implements the VFPv3 architecture. The VFP coprocessor provides a floating-point computation coprocessor that is fully compliant with the ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic. The VFP coprocessor supports all data-processing instructions and data types in the VFPv3 architecture [2]. VFP coprocessor support a large range of floating point operations such as single-precision and double-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. It also provides hardware support for fixed-point and floating-point conversions.

Since ARM Cortex-A8 is a low-cut model in ARM Cortex™-A series processors, it has a cut-down *VFPLite* module instead of a full VFP module, and require roughly ten times more clock cycles per float operation [3].

5. Memory Management

In this chapter, we compare the memory management of ARM Cortex-A8 processor with Intel-i7.

Cortex-A8

Cortex-A8 can support a two-level cache hierarchy. The memory hierarchy as a whole is shown in the Fig 4. There are several components relating to the memory of Cortex-A8.

1) Memory Management Unit

The MMU works with the L1 and L2 memory system to translate virtual addresses to physical addresses. It also controls accesses to and from external memory. Memory management is handled by a pair of TLBs (I and D), each of which are fully associative with 32 entries and a variable page size (4 KB, 16 KB, 64KB, 1 MB, and 16 MB). Each TLB entry contains a virtual address, a page size, a physical address, and a set of memory attributes.

When the processor generates a memory access, the MMU first performs a lookup for the requested virtual address and current ASID and security state in the relevant instruction or data TLB. If the lookup in the previous step misses, the MMU performs a hardware translation table walk.

If the MMU finds a matching TLB entry, it first determines if the access is enabled with the access permission bits and the domain. If the matching entry does not pass the permission checks, the MMU

signals a memory abort. Second it determines if the access is secure or nonsecure, cached or noncached, and device or shared. After then, the MMU translates the virtual address to a physical address for the memory access.

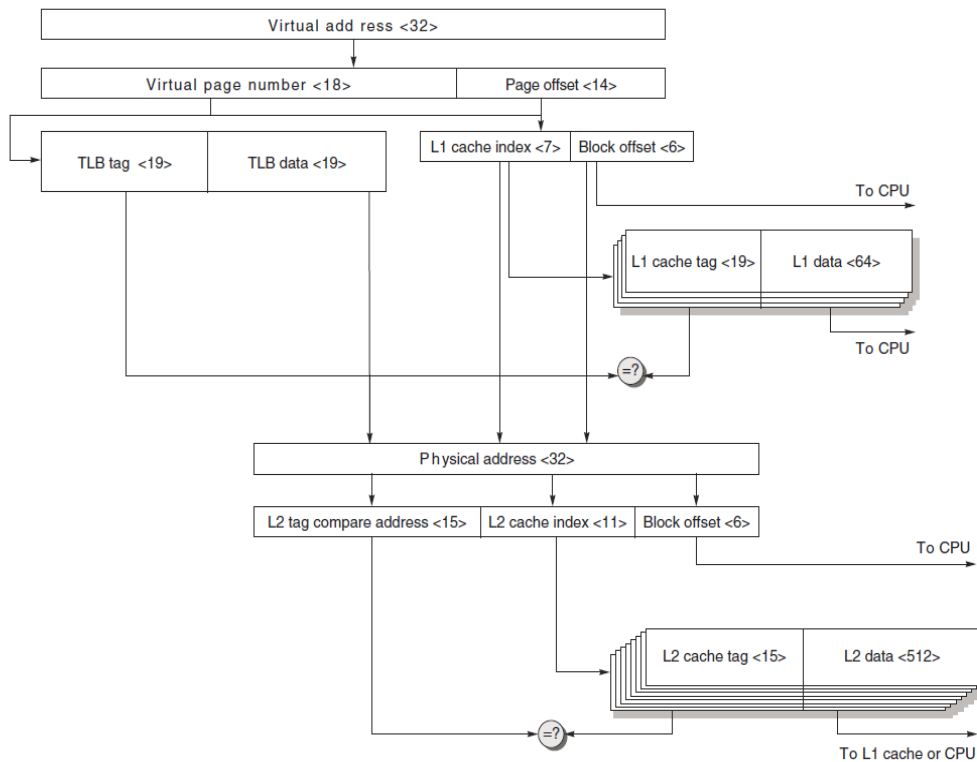


Fig 4. The memory hierarchy

2) Level 1 Memory System

The level 1 caches consist of a pair of caches (for Instruction & Data), each of which is 4-way set associative of configurable size of 16KB or 32KB. They are physically tagged, and virtually indexed for instruction and physically indexed for data. Both the instruction cache and the data cache are capable of providing two words per cycle for all requesting sources. Data cache can provide four words per cycle for NEON or VFP memory accesses. When a read access is not present in the cache, a cache miss comes and a critical word-first cache refilling performed by the caches.

The level 1 caches also implements the feature of parity error detection. Its purpose is to increase the tolerance to memory faults. The L1 memory system instruction and data caches support parity detection on data arrays. There is one parity bit for each data byte. For data cache, because the dirty bit is also held in the data array, there is a corresponding parity bit to cover the dirty bit. Parity errors

reported by instruction cache accesses result in precise prefetch aborts. Parity errors reported by data cache accesses result in imprecise data aborts.

3) Level 2 Memory System

The level 2 memory system is 8-way set associative of configurable size of 0KB, 128KB, 256KB, 512KB, and 1MB. It is tightly coupled to the L1 data cache and L1 instruction cache. The L2 memory system does not support hardware cache coherency, therefore software intervention is required to maintain coherency in the system.

The L2 cache is partitioned into multiple banks to enable parallel operations. There are two levels of banking. The first one is that the tag array is partitioned into multiple banks to enable up to two requests to access different tag banks of the L2 cache simultaneously. The second level is that each tag bank is partitioned into multiple data banks to enable streaming accesses to the data banks. Its structure is shown in Fig 5.

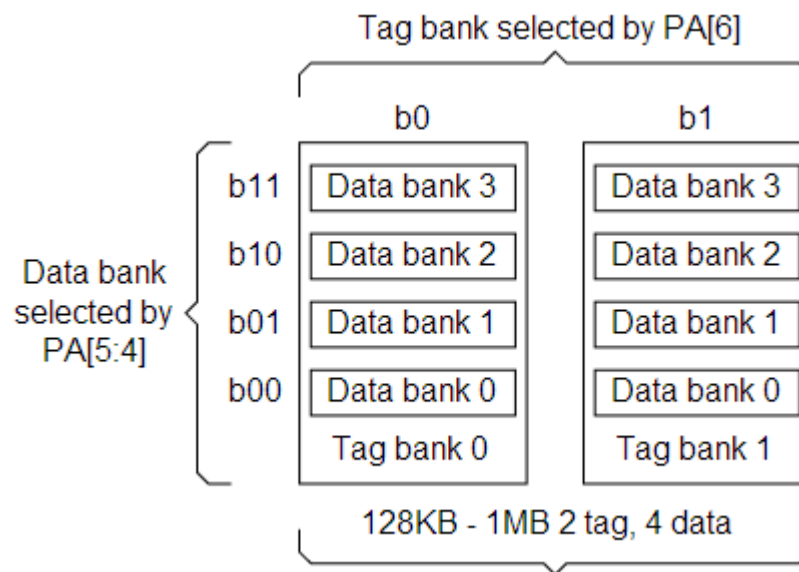


Fig 5. Structure of L2 cache

The L2 cache controller supports transactions from a programmable preloading engine (PLE). The L2 PLE has two channels to permit two blocks of data movement to or from the L2 cache RAM. The L2 PLE also supports the ability to lock data to a specific L2 cache way. If software requires the data to always remain resident in the L2 cache way, software can lock the specific cache way per channel when the PLE transfers data to or from the L2 cache RAM. Locking of a specified way only guarantees that the PLE is within the L2 cache RAM after completion. If the way is not locked, it is possible that the software might have evicted or replaced data with the way that the PLE is transferring data.

The L2 memory supports parity detection on the tag arrays. The data arrays can support parity or Error Correction Code (ECC). If ECC support is implemented, two extra cycles are added to the L2 pipeline to perform the checking and correction functionality. In addition, ECC introduces extra cycles to support read-modified-write conditions when a subset of the data covered by the ECC logic is updated. The ECC supports single-bit correction and double-bit detection.

4) Performance

The memory hierarchy of the Cortex-A8 was simulated with 32 KB primary caches and a 1 MB eight-way set associative L2 cache using the integer Minnespec benchmarks. Minnespec is a set of benchmarks consisting of the SPEC2000 benchmarks but with different inputs that reduce the running times by several orders of magnitude. Although the use of smaller inputs does not change the instruction mix, it does affect the cache behavior.

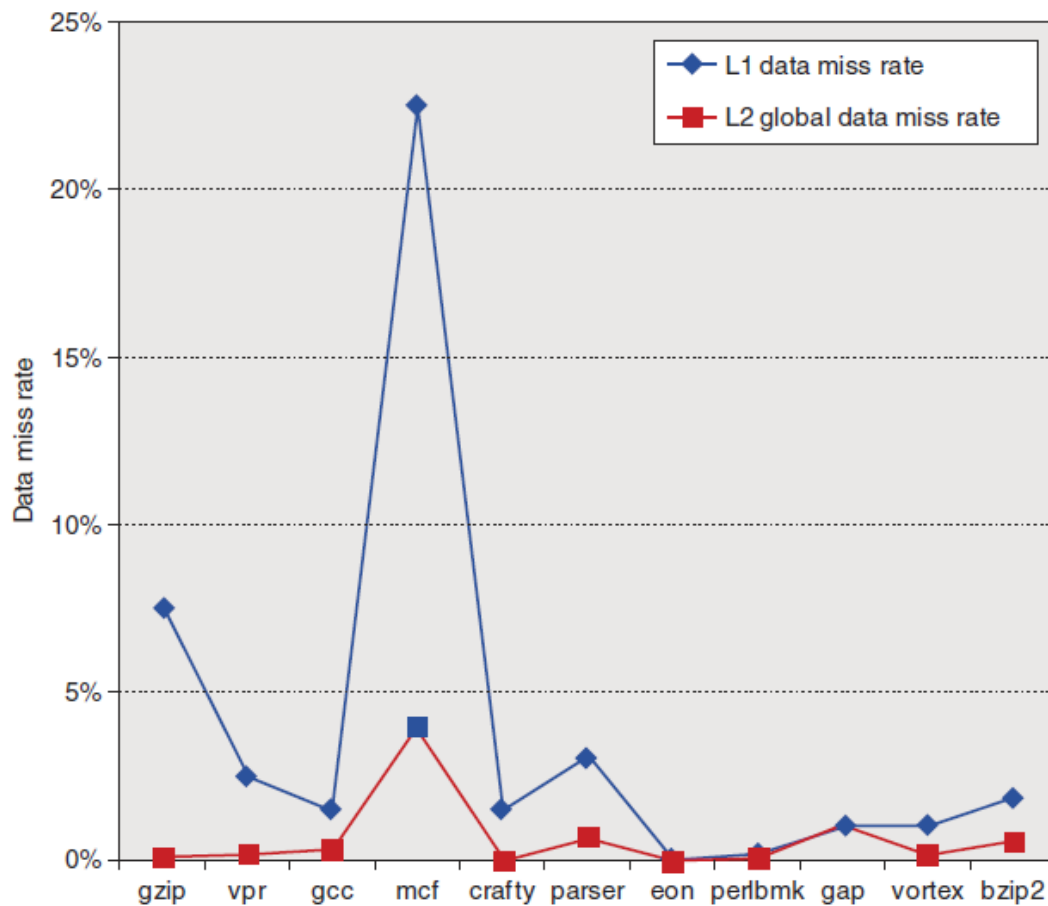


Fig 6. Data cache misses

The instruction cache miss rates for these benchmarks (and also for the full SPEC2000 versions on which Minnespec is based) are very small even for just the L1: close to zero for most and under 1% for

all of them. This low rate probably results from the computationally intensive nature of the SPEC programs and the four-way set associative cache that eliminates most conflict misses.

For the data cache test, there are significant L1 and L2 miss rates. Fig 6 illustrates the data cache misses.

Intel-i7

1) Memory-hierarchy

The cache hierarchy contains a first level instruction cache, a first level data cache (L1 DCache) and a second level (L2) cache, in each core. The L1D cache may be shared by two logical processors if the processor support Intel HyperThreading Technology. The L2 cache is shared by instructions and data. All cores in a physical processor package connect to a shared last level cache (LLC) via a ring connection. The caches use the services of the Instruction Translation Lookaside Buffer (ITLB), Data Translation Lookaside Buffer (DTLB) and Shared Translation Lookaside Buffer (STLB) to translate linear addresses to physical address.

The i7 can support up to three memory channels, each consisting of a separate set of DIMMs, and each of which can transfer in parallel. Using DDR3-1066(DIMM PC8500), the i7 has a peak memory bandwidth of just over 25 GB/sec. i7 uses 48-bit virtual addresses and 36-bit physical addresses, yielding a maximum physical memory of 36 GB.

The memory and the TLB structure of Intel i7 is listed as Fig 7 below:

Characteristic	L1	L2	L3
Size	32 KB I/32 KB D	256 KB	2 MB per core
Associativity	4-way I/8-way D	8-way	16-way
Access latency	4 cycles, pipelined	10 cycles	35 cycles
Replacement scheme	Pseudo-LRU	Pseudo-LRU	Pseudo-LRU but with an ordered selection algorithm

Characteristic	Instruction TLB	Data DLB	Second-level TLB
Size	128	64	512
Associativity	4-way	4-way	4-way
Replacement	Pseudo-LRU	Pseudo-LRU	Pseudo-LRU
Access latency	1 cycle	1 cycle	6 cycles
Miss	7 cycles	7 cycles	Hundreds of cycles to access page table

Fig 7. The memory and the TLB structure of Intel i7

Since it is a multicore processor, the system-on-a-chip design provides a high bandwidth bi-directional ring bus to connect between the IA cores and various sub-systems in the uncore.

2) Loads

When an instruction reads data from a memory location that has write-back (WB) type, the processor looks for it in the caches and memory. The access lookup order and best case latency are listed in Fig 8.

Level	Latency (cycles)	Bandwidth (per core per cycle)
L1 Data	4	2 x16 bytes
L2 (Unified)	12	1 x 32 bytes
Third Level (LLC)	26-31	1 x 32 bytes
L2 and L1 DCache in other cores if applicable	43- clean hit; 60 - dirty hit	

Fig 8. The access lookup order and best case latency

The actual latency can vary depending on the cache queue occupancy, LLC ring occupancy, memory components, and their parameters.

The LLC is inclusive of all cache levels above it - data contained in the core caches must also reside in the LLC. Each cache line in the LLC holds an indication of the cores that may have this line in their L2 and L1 caches. If there is an indication in the LLC that other cores may hold the line of interest and its state might have to modify, there is a lookup into the L1 DCache and L2 of these cores too. The lookup is called "clean" if it does not require fetching data from the other core caches. The lookup is called "dirty" if modified data has to be fetched Stores

When an instruction writes data to a memory location that has a write back memory type, the processor first ensures that it has the line containing this memory location in its L1 DCache, in

Exclusive or Modified MESI state. If the cache line is not there, in the right state, the processor fetches it from the next levels of the memory hierarchy using a Read for Ownership request. Once the cache line is in the L1 DCache, the new data is written to it, and the line is marked as Modified.

3) Level 1 Data Cache

The L1 DCache is the first level data cache. It manages all load and store requests from all types through its internal data structures. It is a write-back write-allocate cache. Stores that hit in the DCU do not update the lower levels of the memory hierarchy. Stores that miss the DCU allocate a cache line.

L1 Data Cache Components

Component	Intel microarchitecture code name Sandy Bridge	Intel microarchitecture code name Nehalem
Data Cache Unit (DCU)	32KB, 8 ways	32KB, 8 ways
Load buffers	64 entries	48 entries
Store buffers	36 entries	32 entries
Line fill buffers (LFB)	10 entries	10 entries

Fig 9. L1 Data Cache Components

The L1 DCache maintains requests which cannot be serviced immediately to completion. Some reasons for requests that are delayed: cache misses, unaligned access that splits across cache lines, data not ready to be forwarded from a preceding store, loads experiencing bank collisions, and load block due to cache line replacement.

4) Address Translation

The DTLB can perform three linear to physical address translations every cycle, two for load addresses and one for a store address. If the address is missing in the DTLB, the processor looks for it in the STLB, which holds data and instruction address translations. The penalty of a DTLB miss that hits the STLB is seven cycles.

5) Store Forwarding

If a load follows a store and reloads the data that the store writes to memory, the data can forward directly from the store operation to the load. This process, called store to load forwarding, saves cycles by enabling the load to obtain the data directly from the store operation instead of through memory.

6) Memory Disambiguation

A load operation may depend on a preceding store. Many microarchitectures block loads until all

preceding store addresses are known. The memory disambiguator predicts which loads will not depend on any previous stores. When the disambiguator predicts that a load does not have such a dependency, the load takes its data from the L1 data cache even when the store address is unknown. This hides the load latency. Eventually, the prediction is verified. If an actual conflict is detected, the load and all succeeding instructions are re-executed.

7) Data Prefetching

Data can be speculatively loaded to the L1 DCache using software prefetching, hardware prefetching, or any combination of the two. The goal of the prefetchers is to automatically predict which data the program is about to consume. If this data is not close-by to the execution core or inner cache, the prefetchers bring it from the next levels of cache hierarchy and memory. Prefetching has the following effects:

- Improves performance if data is arranged sequentially in the order used in the program.
- May cause slight performance degradation due to bandwidth issues, if access patterns are sparse instead of local.
- On rare occasions, if the algorithm's working set is tuned to occupy most of the cache and unneeded prefetches evict lines required by the program, hardware prefetcher may cause severe performance degradation due to cache capacity of L1.

6. Conclusion

In this project, we deep into the structure of ARM Cortex-A* series processors. Taking the market-proven ARM Cortex-A8 processor for instance, we analyze the structure from memory management, instruction sets, enhanced ARM exclusive features and other point of views. We have a better understanding of the structure of the most popular processor - ARM Cortex-A* series processors.

7. References

- [1] ARM architecture: <http://www.arm.com/products/processors/technologies/neon.php>
- [2] ARM Limited. "Cortex™-A8 Technical Reference Manual." Revision: r2p1, 16 Nov., 2007
- [3] Wikipedia. "ARM architecture." http://en.wikipedia.org/wiki/ARM_architecture
- [4] ARM Limited. "Cortex-A9 Processor". Arm.com. Retrieved 21 November 2011.
- [5] Hennessy, John L., and David A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2012.

- [6] <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.cortexa.a8/index.html>
- [7] <http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf>
- [8] <http://www.arm.com/products/processors/index.php>
- [9] <http://www.arm.com/products/processors/cortex-a/>
- [10] <http://www.arm.com/markets/index.php>
- [11] <http://www.arm.com/products/processors/technologies/biglittleprocessing.php>
- [12] http://en.wikipedia.org/wiki/ARM_Cortex-A