University of Moratuwa
EN 2031
Fundamental of Computer Organization& Design

Semester III

Processor Dissection

(INTEL Corei7 – 12th Generation & Cortex 510A)

Group Alder Lake
Group-2

Name	Index No.	Area
Gunawardana W. N. M	210199D	
Rajapaksha I.P.D.D	210503H	Instruction Set Architecture, Micro-Architecture Arithmetic & Logic Functions of Cortex 510A
Dilshan N.L.	210129P	Cache Memory and Memory Interfacing, Timing Related to Memory for Cortex-A510, micro-architecture for Intel Core i7-12th Generation and comparison.

October 22, 2023

Table of Contents

1. INTEL Corei7 – 12th Generation

1.1. Introduction

The 12th Generation Intel Core Alder Lake processor represents a major leap in Intel's CPU architecture. It is the cornerstone of the upcoming 12th-generation Core processors, offering a blend of cutting-edge features and innovations. With a hybrid computing architecture, combining powerful large cores and efficient small cores, along with advancements in microarchitecture, manufacturing technology, and memory support, Alder Lake promises to deliver substantial gains in both performance and energy efficiency. In this introduction, we will explore the core attributes and features of this highly anticipated processor architecture. These are some of the main features of the processor,

- Cores – 8 performance and 4 Efficiency.
- Avg clock rates – (1.4 – 5.0) GHz
- Smart cache – 25 MB
- Power Consumption – (35 – 190) W
- Released date – NOV 4, 2021
- Average price – (339 – 409) $

1.2. Instruction Set Architecture of the Processor.

1.2.1. Introduction

The 12th Generation Intel Core i7 processors are based on the x86-64 instruction set architecture. The x86-64 architecture is an extension of the original x86 architecture, which has been a prevalent and widely used architecture for several decades in the world of computing. It is characterized by its complex instruction set computer (CISC) design, offering a broad and versatile set of instructions for performing various operations. This instruction set architecture is backward-compatible with its predecessors, allowing software written for previous generations of x86 processors to run on the 12th Gen Intel Core i7 processors. It also features 64-bit support, enabling efficient handling of large memory spaces and enhanced performance for 64-bit applications. Intel 64 architecture contains two sub-modes,

Compatibility Mode: This sub-mode allows old 16-bit and 32-bit apps to work flawlessly in a 64-bit OS, reducing the need for re-compilation. It offers an environment like 32-bit protected mode, with access to the first 4 GB of linear address space. Compatibility mode supports all privilege levels and permits access to physical memory beyond 4 GB via PAE. However, it does not handle apps in Virtual 8086 mode or those employing hardware task management.

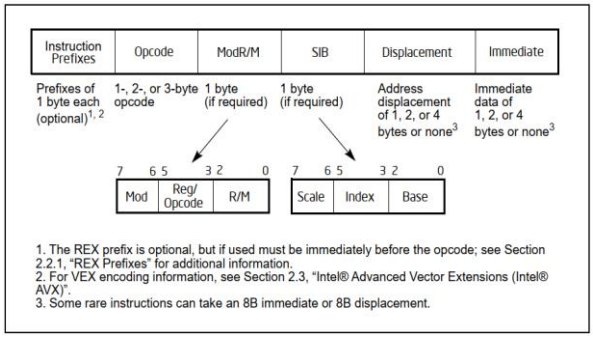
64-bit Mode: In this sub-mode, a 64-bit OS may execute applications intended for 64-bit linear address space. It boosts the processor's capabilities, increasing the number of general-purpose and SIMD registers. These registers are extended to 64 bits, and the REX opcode prefix is introduced for accessing register extensions. By default, it works with a 64-bit address size and a 32-bit operand size, although this may be modified on a per-instruction basis using REX prefixes. This mode unlocks the full power of the CPU, allowing the usage of 64-bit registers and addresses in a backward-compatible manner.

1.2.2 Instruction Classes and Instruction Formats

The x86-64 instruction set architecture is employed by Intel Alder Lake CPUs, with instruction formats of varying lengths determined by specific operations and operands. This simplified overview allows for clear identification and understanding of the essential elements of x86-64 instruction formats.

1. **Opcode (Operation Code):** The opcode specifies the operation to be performed, such as addition or subtraction. It's essential for the instruction's behavior and functionality, making it vital in executing instructions within the system.
2. **Prefixes:** Prefixes may be utilized to modify or provide additional information to instructions. These prefixes can indicate various elements such as the size of the data or whether an operation should affect flags.
3. **ModR/M Byte:** The ModR/M byte specifies operands as registers, memory locations, or both.
4. **SIB (Scale Index Base) Byte:** SIB byte is used in some instruction sets for complex memory addressing modes with scaling factors and index registers.
5. **Displacement:** When the instruction is designed to work with memory operands, a displacement value may be provided to indicate the specific memory address.
6. **Immediate Data:** Immediate data is a constant value that serves as an operand in many instructions. The placement and size of the immediate data in the instruction format can differ.
7. **Registers:** Registers may serve as operands, depending on the operation being performed. The ModR/M byte specifies the registers involved in the operation.

The format and encoding of instructions can vary depending on the operation and addressing mode. x86-64 architecture is complex due to its historical development and backward compatibility. For up-to-date information on Intel Alder Lake CPUs.



Source: Intel® 64 and IA-32 Architectures Software Developer's Manual

1.2.3. Instruction Set

The x86-64 instruction set is quite extensive and versatile, and it includes many instructions. Some of the key features of this instruction set include:

- General-Purpose Register.
- Memory Addressing Modes.
- SIMD (Single Instruction, Multiple Data.
- Floating-Point Instructions.
- Control Flow Instructions.
- String and Memory Manipulation.
- System and I/O Instructions.
- Vector Extensions (AVX and AVX-512).
- Security Extensions (AES-NI, SHA Extension).
- Virtualization Support (VT-x).

Instruction Syntax

Data Movement instructions.

- MOV: transfer data between registers or between registers and memory
- LEA: compute the effective address of a memory operand and store it in a register
- XCHG: exchange the values of two operands, typically registers.
- PUSH: push data onto the stack
- POP: retrieve data from the stack
- MOVS: move data between memory locations in string operations.
- LODS: load a value from a memory location into a register in string operations
- STOS (store string): store data from a register into a memory location in string operations.

Arithmetic and Logic Instructions

- ADD: addition
- SUB: subtraction
- MUL: unsigned multiplication
- DIV: unsigned division
- AND: bitwise AND
- OR: bitwise OR
- XOR: bitwise XOR (exclusive OR)
- NOT: bitwise NOT
- SHR: logical right shift
- SHL: logical left shift
- SAL: arithmetic left shift
- SAR: arithmetic right shift

Control Flow Instructions

- JMP (Jump): unconditional branching to change program flow.
- CALL (Call Procedure): Invokes subroutines, saving return addresses.
- RET (Return from Procedure): Resumes execution at the caller's location.
- JZ (Jump if Zero): Jumps when the result is zero.
- JNZ (Jump if Not Zero): Jumps when the result is not zero.
- JG (Jump if Greater): Jumps when the first operand is greater.

String and Memory Instructions

- REP: Repeats string operations.
- CMPS: Compares strings.
- SCAS: Scans for a value in a string.
- MOVS: Moves data in string operations.
- LODS: Loads data from memory.
- STOS: Stores data in a string operation.

Conditional Instructions

- CMP: Compares values.
- TEST: Performs bitwise logical tests.
- SETcc: Sets a destination operand based on a condition.
- CMOVcc: Moves data conditionally based on a specified condition.

Segment and Memory Management Instructions

- MOV (Segment Registers): Moves data to/from segment registers.
- LDMXCSR: Loads the MXCSR register.
- STMXCSR: Stores the MXCSR register.

Floating-Point and SIMD Instructions

- FADD: Performs floating-point addition.
- FMUL: Performs floating-point multiplication.
- FCOM: Compares floating-point values.

- FSUB: Performs floating-point subtraction.
- FLD (Load Floating-Point Value): Loads a floating-point value onto the FPU (Floating-Point Unit) stack.
- FSTP (Store Floating-Point Value): Stores a floating-point value from the FPU stack to memory or a register.

System and Privileged Instructions

- HLT: Halts the processor.
- IN: Inputs data from a port.
- OUT: Outputs data to a port.
- CPUID: Returns processor identification and feature information.
- RDTSC: Reads the time-stamp counter for performance measurement.

I/O Instructions

- IN: Inputs data from an I/O port.
- OUT: Outputs data to an I/O port.
- INS: Inputs data from an I/O port to memory.
- OUTS: Outputs data from memory to an I/O port.

String Comparison and Search Instructions

- CMPS: Compares strings.
- SCAS: Scans for a value in a string.
- LODS: Loads data from a string.
- STOS: Stores data into a string.

Bit Manipulation Instructions

- BT (Bit Test): Tests a specific bit in a register or memory location.
- BTS (Bit Test and Set): Tests a bit and sets it if it's clear.
- BTR (Bit Test and Reset): Tests a bit and resets it if it's set.
- BSF (Bit Scan Forward): Scans for the position of the first set (1) bit.
- BSR (Bit Scan Reverse): Scans for the position of the most significant set (1) bit.

Security and Cryptographic Instructions

AES-NI (Advanced Encryption Standard New Instructions):

- AES-ENCRYPT: Performs AES encryption on data.
- AES-DECRYPT: Performs AES decryption on data.

SHA (Secure Hash Algorithm) Instructions:

- SHA-1: Performs SHA-1 hashing.
- SHA-256: Performs SHA-256 hashing.
- SHA-512: Performs SHA-512 hashing.

1.3. Micro-Architecture

Alder Lake is Intel's 12th Gen Core processor family with a hybrid architecture. It features both Performance (P-core) and Efficiency (E-core) cores. The P-cores are based on the Golden Cove microarchitecture (for single-threaded performance and low latency) and the E-cores are based on the Gracemont architecture (for power efficiency and multi-threaded workloads).

Golden Cove microarchitecture:

- Improved branch prediction
- Larger out-of-order execution window
- Enhanced instruction scheduling

Gracemont microarchitecture:

- Focused on power efficiency.
- Wide execution pipeline
- Efficient instruction decode.

The processor designed for desktops has up to 16 cores (8 P-cores + 8 E-cores) and 24 threads. Also, it has a non-inclusive last-level cache (L3 cache) of up to 30MB. The processor supports DDR5, DDR4, LPDDR5, and LPDDR4X memory. It has 20 lanes of PCIe, including 16 lanes of PCIe 5.0 and 4 lanes of PCIe 4.0. Alder Lake features a dual-bandwidth ring, capable of 1000 GB/s of bandwidth. Also, the intel thread director is a key feature for task management.

1.4. ALU Functions

The principal role of the ALU (Arithmetic Logic Unit) in a CPU (Central Processing Unit) is to conduct arithmetic and logic operations on data. The ALU is a fundamental component of the CPU responsible for conducting numerous operations, including:

- Arithmetic Operations: The ALU performs fundamental arithmetic operations such as addition, subtraction, multiplication, and division. These activities are fundamental for mathematical computations and numerical processing in computer applications.
- Logic Operations: The ALU conducts logical operations such as AND, OR, XOR, and NOT. These procedures are critical for making judgments, comparing data, and managing the flow of program execution.
- Data Comparison: The ALU compares data to discover connections such as equality, inequality, more than, or less than. These comparisons are employed in conditional branching and decision-making within programs.

- Bitwise Operations: Bitwise operations, including shifting and rotating bits, are accomplished by the ALU. These operations are used for manipulating binary data, setting, or clearing particular bits, and data transformation.
- Overflow Detection: The ALU may detect arithmetic overflow, a condition that happens when the result of an operation cannot be expressed within the available number of bits. Overflow detection is vital for assuring the correctness of calculations.
- Conditional Operations: The ALU can perform operations conditionally dependent on the status of condition code flags. For instance, it can conduct conditional addition only if a given condition is satisfied.
- Microcode Execution: In certain CPUs, the ALU executes microcode instructions, which are a collection of low-level instructions used to accomplish complicated tasks and govern the processor's behavior.

The ALU's capacity to conduct these operations promptly and efficiently is important to the CPU's ability to execute instructions, analyze data, and carry out the functions required by computer programs. The design and capabilities of the ALU might vary across different CPU architectures, but its essential purpose of conducting arithmetic and logic operations stays consistent.

1.5. Cache Memory and Memory Interfacing

Cache Memory:

Cache memory is a crucial component of CPUs that significantly improves processing performance by lowering memory access latency. Different types of cache memory are used in Intel 64 and IA-32 architectures, including L1, L2, and L3 caches. These caches help reduce the time required to retrieve frequently used instructions and data. CPUID instruction is used to identify cache sizes and attributes for a specific processor.

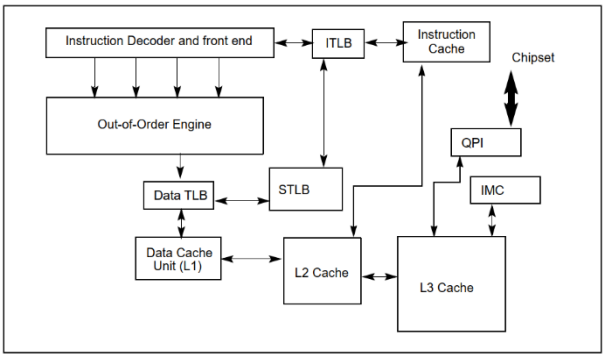


Figure: Cache structure of I7 processors.

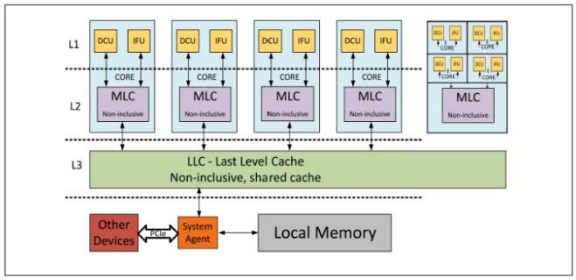


Figure: Smart Cache Technology

Intel Smart Cache Technology is a fundamental component of 12th-generation Intel Core CPUs, boosting memory efficiency. This technique offers a shared Last Level Cache (LLC), referred to as the 3rd level cache, which is non-inclusive. The LLC is shared by IA cores and the Processor Graphics unit, allowing data transfer.

For P Cores, both 1st and 2nd-level caches are private, facilitating isolation. The 1st level cache comprises a 48KB data cache and a 32KB instruction cache, both 12-way associative. E Cores have non-shared 1st level caches with a 32KB data cache and a 64KB instruction cache, both 8-way associative.

The 2nd level cache, commonly termed mid-level cache, is where data and instructions live. P Cores include a 1.25MB 2nd level cache, 10-way non-inclusive, whereas 4 E Cores share a 2MB 2nd level cache, which is 16-way non-inclusive. Intel Smart Cache Technology improves memory access, balancing speed, and efficiency for varied core types.

Memory Interfacing:

Memory interfacing is a crucial aspect of CPU design, as it determines how the processor interacts with system memory. IA-32 CPUs, including Intel Core i7 processors, use caching techniques to improve efficiency. Various memory types, such as Write-Back and Write Combining, are available for optimizing cache utilization. Understanding cache behavior, size, and memory types can enhance program performance.

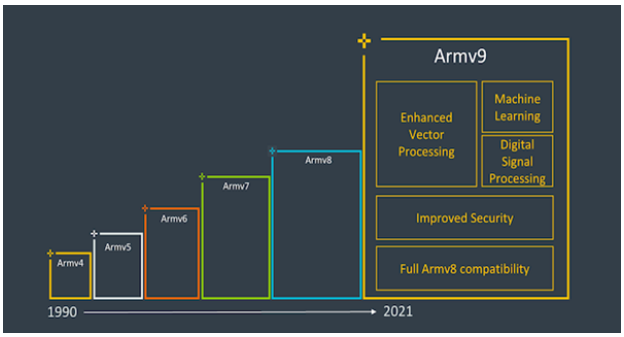
2.Cortex-A510

2.1. Introduction – Cortex-A510

The ARM Cortex-A510, which was introduced in 2021, is a revolutionary processor that emphasizes excellent efficiency. It is the first of its kind under the ARMv9 architecture. This "LITTLE" CPU core was meticulously developed by the Cambridge design team at ARM Holdings as the Cortex-A55's replacement. It is a brand-new 64-bit CPU concept. The Cortex-A510 has shown to be a worthy complement to the powerful Cortex-A710

"big" core, increasing processing performance all around. This microarchitecture, code-named "Klein," offers major advancements. In June 2022, ARM announced an update for the Cortex-A510 CPU core that would further improve its power efficiency by 5% and expand its scalability from up to 8 cores to a remarkable 12 cores. This evolution cements the Cortex-A510 as a pivotal player in the realm of high-efficiency processing. In this report, we will dissect and delve into the core components and functioning of the Cortex-A510 processor, shedding light on its innovative features and contributions to the world of computing.

2.2. Instruction Set Architecture



The ARM Cortex-A510 is a processor that runs on the AArch64 execution state and is based on the A64 Instruction Set Architecture. Nevertheless, because of its backward compatibility, it can process using the A32 and T32 architectures, albeit only at EL0. Improvements were made as a result of the introduction of the A64 instruction set. Some of them are,

1. Facilitating Distant Instructions

This archive is because of a port for reaching instructions located at a significant distance from the current instruction, aiding in the execution of code spread across memory.

2. Enhanced Encoding Scheme Consistency

The A32 architecture suffered from issues related to efficiency due to its lack of a fixed encoding scheme. The A64 architecture resolves this problem by maintaining a consistent encoding scheme, leading to improved efficiency.

3. Versatile Constants

The A64 instruction set is tailored to offer a wide range of constant options specific to various data types. Different instructions can accept different types of constants. For instance, arithmetic instructions can handle 12-bit immediate constants.

4. User-Friendly Data Types

The A64 instruction set simplifies operations, especially for users of languages like C and Java, by seamlessly handling 64-bit signed and unsigned data types.

2.2.1. Memory Access Instructions

Load Instructions

- LDRB: Load a byte and zero-extend it.
- LDRSB: Load a signed byte and sign-extend it.
- LDRH: Load a halfword and zero-extend it.
- LDRSH: Load a signed halfword and sign-extend it.
- LDRSW: Load a signed word and sign-extend it

Store Instructions:

- STR: Save the contents of a register.
- STRH: Save a halfword.
- STRB: Save a byte.

2.2.2. Data processing instructions

Arithmetic Instructions:

- ADD: Perform a register-based addition.
- SUB: Execute a register-based subtraction.
- ADC: Add with carry (register).
- SBC: Subtract with carry (register).
- NEG: Negate (register).

Logical Instructions:

- AND: Perform a bitwise AND operation (register).
- BIC: Clear bits in a register.
- ORR: Execute a bitwise inclusive OR operation (register).
- ORN: Execute a bitwise inclusive OR NOT operation (register).
- EOR: Perform a bitwise exclusive OR operation (register).
- EON: Execute a bitwise exclusive OR NOT operation (register).

Comparison Instructions:

- CMP: Compare using an immediate value.
- CMN: Compare negatively using an immediate value.
- TST: Perform a bitwise test using an immediate value.

Move Instructions:

- MOV: Move data within registers.
- MVN: Move data with a bitwise NOT operation (register).

Multiply Instructions:

- MADD: Multiply and add.

- MNEG: Multiply and negate.
- MSUB: Multiply and subtract.
- MUL: Multiply.
- SMADDL: Signed multiply-add long.
- SMNEGL: Signed multiply-negate long.
- SMSUBL: Signed multiply-subtract long.
- SMULH: Signed multiply returning high half.
- SMULL: Signed multiply long.
- UMADDL: Unsigned multiply-add long.
- UMNEGL: Unsigned multiply-negate long.
- UMSUBL: Unsigned multiply-subtract long.
- UMULH: Unsigned multiply returning high half.
- UMULL: Unsigned multiply long.

Divide Instructions:

- SDIV: Signed divide.
- UDIV: Unsigned divide.

Shift Instructions:

- ASR: Perform an arithmetic shift right.
- LSL: Execute a logical shift left.
- LSR: Perform a logical shift right.
- ROR: Rotate right.

Bitfield and Byte Manipulation Instructions:

- UXTB: Extend the last significant unsigned byte.
- UXTH: Extend the last significant unsigned halfword.
- SXTW: Extend the last significant signed word.
- SXTB: Extend the last significant signed byte.
- SXTH: Extend the last significant signed halfword.
- BFI: Insert a bit field.
- BFM: Move a bit field.
- RBIT: Reverse all bits.
- REV: Reverse the byte order.

Conditional Instructions:

- CSEL: Conditional select, returning the source.
- CSINC: Conditional select, returning source 1 or an incremented value.
- CSINV: Conditional select, returning source 1 or an inverted value.
- CSNEG: Conditional select, returning source 1 or a negated value.
- CSET: Conditional set.
- CSETM: Conditional set mask.
- CINC: Conditional increment.

2.2.3. Flow Control Instructions

Unconditional Branch Instructions:

- B(offset): Perform a relative branch either forward or backward.
- BL(offset): Store the return address in X30 and provide a hint to branch prediction logic.
- BR Xn: Execute an absolute branch to the address stored in register Xn.
- BLR Xn: Store the return address in register Xn and offer a hint to branch prediction logic.
- RET: Perform a return operation.

Conditional Branch Instructions:

- CBZ Rt, label: Compare and branch if the value in register Rt is zero.
- CBNZ Rt, label: Compare and branch if the value in register Rt is non-zero.
- TBNZ Rt, bit, label: Test a bit and branch if it's zero.
- TBNZ Rt, bit, label: Test a bit and branch if it's non-zero.

2.2.4.System Control and Other Instructions

Exception Handling Instructions:

- SVC: Supervisor call, enabling applications to invoke the kernel.
- HVC: Hypervisor call, allowing OS code to invoke the hypervisor.
- SMC: Secure Monitor call, enabling OS or hypervisor to invoke the secure monitor.
- ERET: Return from an exception.

System Register Access Instructions:

- MRS Xt, system register: Copy the value of a system register into a general-purpose register.
- MSR, system register, Xt: Copy the value of a general-purpose register into a system register.

2.2.5 Debug Instructions

- BRK: Enter monitor debug mode.
- HLT: Enter halt debug mode.

Hint Instructions:

- NOP: No operation.
- YIELD: Indicate that the current thread can be swapped out.
- WFE: Wait for an event.
- WFI: Wait for an interrupt.
- SEV: Send an event.
- SEVL: Send a local event.

Floating-Point Instructions:

- FABS: Calculate the absolute value of a floating-point number.

- FNEG: Return the negative value of a floating-point number.
- FSQRT: Return the square root of a floating-point number.
- FADD: Return the sum of two floating-point numbers.
- FSUB: Return the result of subtracting one floating-point number from another.
- FDIV: Perform floating-point division.
- FMUL: Multiply floating-point numbers.
- FNMUL: Multiply and negate the result (fused operation).
- FMADD: Multiply and then add (fused operation).
- FNMSUB: Multiply, negate, and then subtract (fused operation).
- FNMADD: Multiply, negate, and then add (fused operation).
- FPINTy: Round off floating-point values.
- FCMP: Compare floating-point values.
- FCCMP: Perform a conditional comparison of floating-point values.
- FCSEL: Conditionally select floating-point values.
- FCVTSty: Convert floating-point values to integers.
- SCVTF: Convert integers to floating-point values.

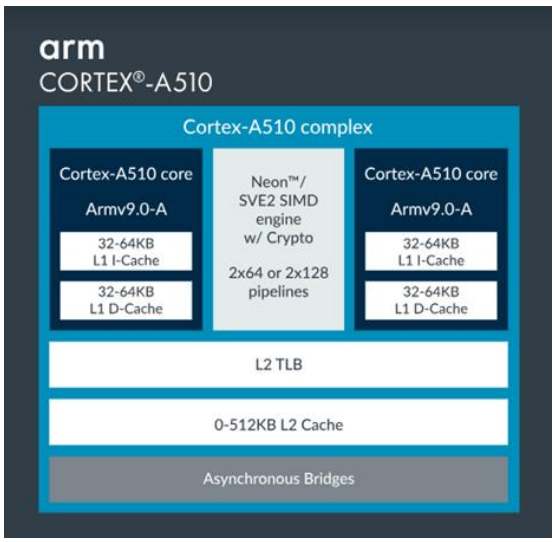
2.2.6.Instruction Classes

There are different classes of the A64 instruction set that share structures and operand positions.

- Data Processing Instructions: Mathematical, logical, comparison, move, multiply, divide, shift, bitfield, byte manipulation, and conditional instructions are all included in this lesson.
- Memory Access Instructions: Comprising load and store instructions.
- Flow Control Instructions: Including instructions for both unconditional and conditional branching
- System Control and Other Instructions: Including floating-point instructions, debugging, hinting, and exception handling.

2.2.7.Instruction Format:

The length of each A64 instruction is always *32 bits. Throughout the instruction set, register fields and immediate values are placed consistently. For 64-bit values in integer registers, "X" names are used, and for 32-bit values, "W" names. Instructions for loading and storing have similar addressing modes. In A64 instructions, access to the program counter and stack pointer is constrained. Different types of instructions*



follow formats.

- Data Processing Instructions Format:

Formats that are frequently used are OPCODE, DESTINATION OPERAND, SOURCE OPERAND 1, and SOURCE OPERAND 2.The types of operands include immediate, register, and shifted register.

- Conditional Instructions Format:

CODE (for example, EQ, NE, or CS), RETURN REGISTER, SOURCE REGISTER, and OPCODE.With this format, flags like N, Z, C, and V are connected.

- Load and Store Instruction Format:

Address for loadings and storing is LDR Rt. Different LOAD instruction OPCODES (Byte, Halfword, Word) load data of various sizes.

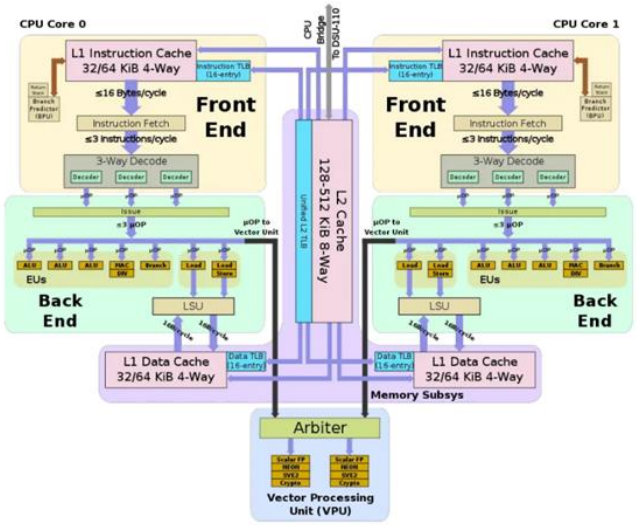
- Flow Control Instruction Format:

The format of both unconditional and conditional instructions is the same: Label, Rt, bit, and OPCODE.

System Control and Other Instructions Format:

OPERANDS, Immediate Values are the instructions for handling exceptions. Operand, general register (system register), and system register (general register) are the system register access instructions. Instructions for debugging: operand, immediate e value. opcode is the hint opcode.

2.3.Micro-Architecture- Cortex-A510



The architecture used by the ARM Cortex-A710 is called ARMv9-A(Harvard). Neon and floating points are both present. The cryptography unit is also an optional inclusion. A CPU cluster can only have a maximum of eight cores. These cores share a single L2 cache that ranges in size from 128 to 512 KBs. It employs a physical addressing mode of 40 bits. Then it has a pipeline that is superscalar and out of order.

2.4 Arithmetic & Logic(ALU) Functions -Cortex-A510

Here are some specifics on the ALU of the Cortex-A510

- All of the common Armv9 integer instructions are supported by the ALU, which is a 32-bit integer ALU.
- The ALU has a cycle capacity of two instructions.
- A five-stage pipeline connects to the ALU.
- A specialized multiplier and divider are built within the ALU.

The ALU of the Cortex-A510 is intended to be a high-performance, low-power ALU that can handle a range of workloads.

2.5. Cache Memory and Memory Interfacing

The processor has L1 instruction cache, L1 data cache and only one L2 cache for both instruction and data.

L1 Instruction Memory System:

- Responsibilities:
  - Fetching instructions and predicting branches.
- Parts:
  - L1 instruction cache
  - L1 Instruction Translation Lookaside Buffer (TLB)
- Feature:
  - 32KB or 64KB L1 instruction cache with 4-way set associativity.
  - This cache has single error detect (SED) parity cache protection.
  - Cache line length: 64 bytes
  - Cache policy: Pseudo-random cache replacement policy

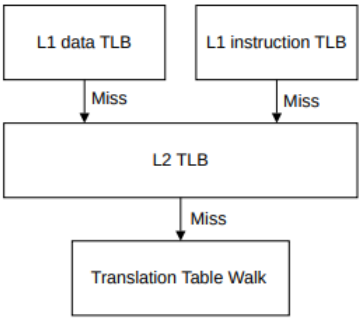
L1 Data Memory System:

- Responsibility:
  - executing load and store instructions and specific instructions like atomics, cache maintenance operations, and memory tagging instructions.
- Parts:
  - L1 data cache
  - L1 data Translation Lookaside Buffer (TLB)
- Features:
  - 32KB or 64KB L1 data cache with 4-way set associativity.
  - Dual 128-bit read and write paths.
  - Cache line length: 64 bytes
  - Data Cache unit (DCU), STore Buffer (STB), Bus Interface Unit (BIU), and Trace and profiling Buffer (BIU)
  - Prefetch engine to detect patterns of cache line requests.

L2 memory system:

- Responsibility:
  - Handling requests from the L1 instruction and data caches, and snoop requests from the L3 memory system
  - Forwarding request from the L3 system
  - Also handles coherent and non-coherent operations from cores and from associated L1 evictions.
- Features:
  - 128KB, 192KB, 256KB, 384KB, or 512KB L2 cache with 8-way set associativity.
  - Cache line length: 64 bytes
  - Dynamic biased cache replacement, Pseudo-exclusive with L1 data caches, Pseudoinclusive with L1 instruction caches for cache policies.

Figure : Translation table walks



2.6. Timing Related to Memory

- Cortex®-A510 core supports hierarchical clock gating.
- Supports Warm and Cold resets.
- Each Cortex®-A510 core has a single clock domain and receives a single clock input.
- An architectural clock gate in the CPU bridge controls the clock input.
- Supports extensive clock gating.

3.0 Comparison between 2 processors

Feature	Intel Core i7-12th Generation	Cortex-A510
Introduction	Hybrid architecture with P-cores and E-cores. Golden Cove and Gracemont microarchitectures. Up to 16 cores and 24 threads. Supports DDR5, DDR4, LPDDR5, and LPDDR4X memory. Dual-bandwidth ring with 1000 GB/s bandwidth. Intel Thread Director for task management.	Emphasizes efficiency. Part of ARMv9 architecture. 64-bit CPU core with "Klein" microarchitecture. Up to 12 cores. Support for L1 and L2 caches.
Instruction Set Architecture	x86-64 ISA with compatibility modes for 16-bit and 32-bit apps. 64-bit support for efficient memory handling. Wide range of instruction classes and formats.	A64 ISA with backward compatibility for A32 and T32. Various data processing, memory access, flow control, and system control instructions.
Micro-Architecture	Hybrid architecture with P-cores and E-cores. Golden Cove microarchitecture for P-cores (single-threaded performance). Gracemont microarchitecture for E-cores (power efficiency). Non-inclusive L3 cache of up to 30MB. Supports various memory types and caching techniques.	ARMv9-A(Harvard) architecture. L2 cache ranging from 128KB to 512KB shared by up to 8 cores. Superscalar, out-of-order pipeline.
ALU Functions	32-bit integer ALU supporting common Armv9 instructions. Two-cycle capacity. Five-stage pipeline. Specialized multiplier and divider. Designed for high-performance and low power consumption.	32-bit integer ALU supporting Armv9 instructions. Two-cycle capacity. Five-stage pipeline. Specialized multiplier and divider. Designed for high-performance with low power consumption.
Cache Memory and Memory Interfacing	L1 instruction and data cache (32KB or 64KB, 4-way set associativity). Non-inclusive L3 cache (up to 30MB). Supports DDR5, DDR4, LPDDR5, LPDDR4X memory types. Multiple L2 cache options (1.25MB for P-cores, 2MB for E-cores).	L1 instruction and data cache (32KB or 64KB, 4-way set associativity). Shared L2 cache (up to 512KB) among multiple cores (max 8). Supports a physical addressing mode of 40 bits.

Timing Related to Memory	Supports hierarchical clock gating. Warm and cold resets. Single clock domain per core. Clock input controlled by architectural clock gate.	Supports extensive clock gating. Hierarchical clock gating. Warm and cold resets. Single clock domain per core.

Reference:

https://en.wikichip.org/wiki/arm_holdings/microarchitectures/cortex-a510#Process_Technology

ARMv9: What is the Big Deal?. What is a Scalable Vector Extension… | by Erik Engheim | Level Up Coding (gitconnected.com)

ARM Cortex-A510 wiki | TheReaderWiki

Arm Announces Mobile Armv9 CPU Microarchitectures: Cortex-X2, Cortex-A710 & Cortex-A510 (anandtech.com)

https://chipsandcheese.com/2023/10/01/arms-cortex-a510-two-kids-in-a-trench-coat/