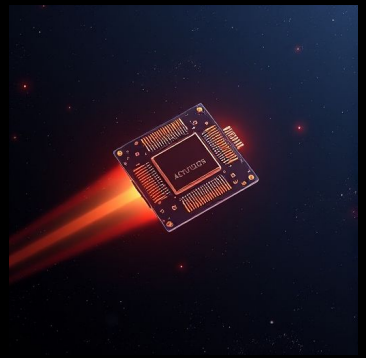
ARM Architecture

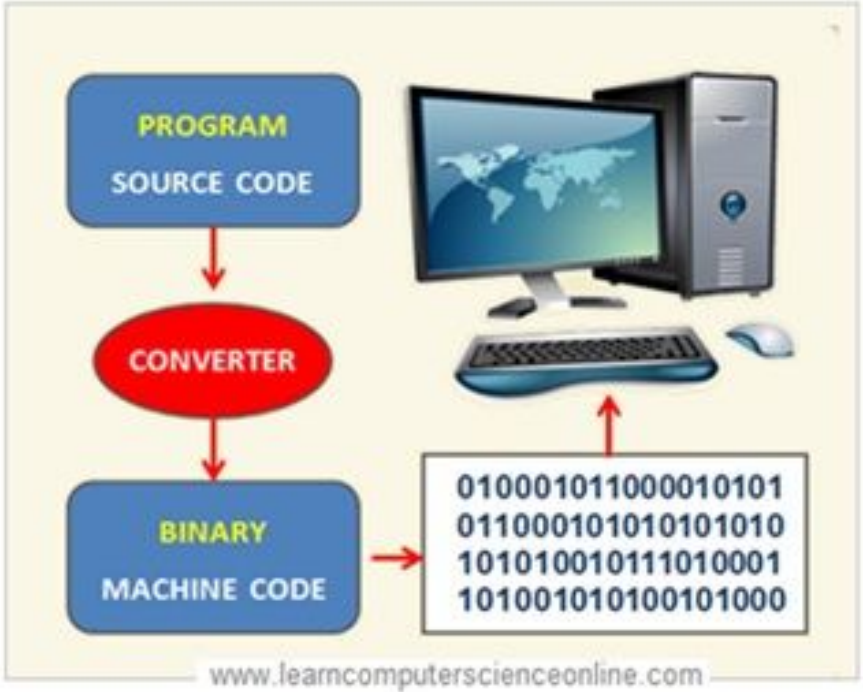


Is RISC and CISC Still an Important Distinction?

High Level Program

```
int main()
{
// Variable declaration
int a, b, sum;
// Take two numbers as input from the
user
scanf("%d %d", &a, &b);
// Add the numbers and assign the value
// to  some variable
sum = a + b;
// Use the calculated value
printf("%d\n", sum);
return 0;
// End of program
}
```

Low Level Machine Instructions



CISC (x86) vs. RISC (ARM) ISA

- Complex Instruction Set Computing(CISC)
  - Has a larger, more complex set of variable-length instructions that can perform multiple operations in a single instruction but often require multiple cycles to execute
  - Intel and AMD (x86)

- Reduced Instruction Set Computing (RISC)
  - Uses a small, simple, fixed-length instruction set designed to execute , with simpler hardware and efficient pipelining.
  - RISCV and ARM

ISA (Instruction Set Architecture)

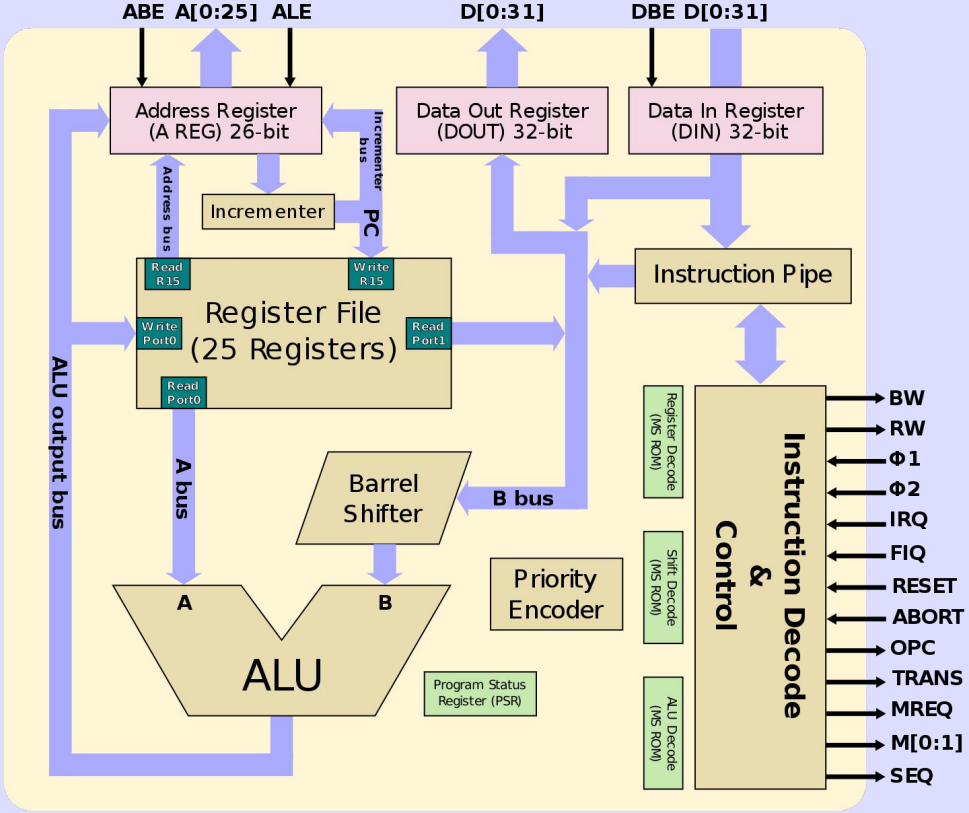
The Set of all instructions a CPU can run. The ISA is a specification.

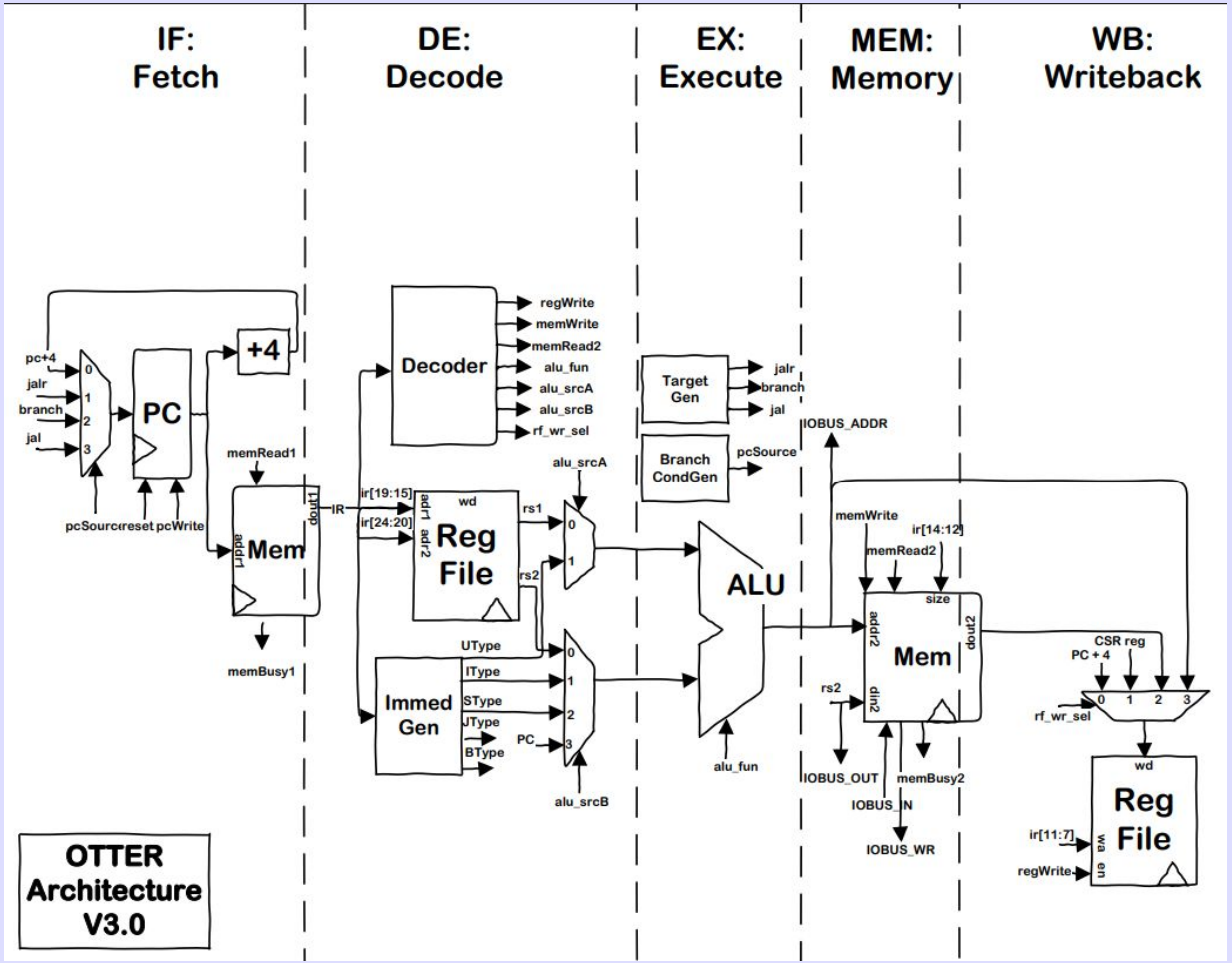
Arm v1

- Introduced in 1985
- 45 instructions 32 bit instructions
- No support for multiplication, or FPUs
- However, the addressing mode uses a 26-bit address space, which combines a 24-bit address with 2 additional bits for processor flags (e.g., mode bits). This limits the directly addressable memory to bytes, or 64 MB
- Movement instructions to move immedients and values in registers
- There's 45 instructions and 23 mnemonics
- Was only used on the Arm1 and a couple of thousand chips were ever created

Arm1

- Introduced in 1985
- 3 um process
- Some instructions take 5 cycles
- 8 million instructions/s
- Limited hardware (single ALU
- Some instructions will take 5 cycles instead of 3
- Store takes 5
- Execute will compute the addr and then data



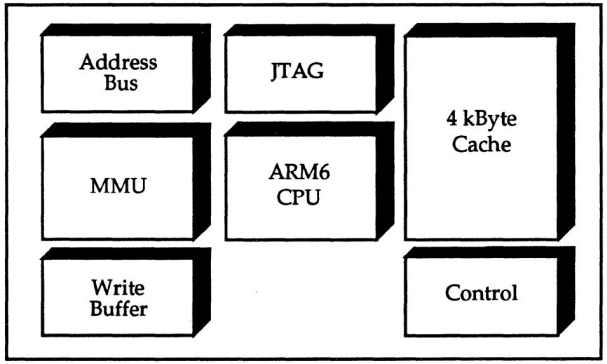

IF:
Fetch
DE:
Decode
EX:
Execute
MEM:
Memory
WB:
Writeback
pc+4
jalr
branch
jal
PC
+4
pcSource
reset
pcWrite
memRead1
Mem
addr1
dout1
memBusy1
IR
ir[19:15]
ir[24:20]
Decoder
regWrite
memWrite
memRead2
alu_fun
alu_srcA
alu_srcB
rf_wr_sel
Reg
File
adr1
adr2
wd
rs1
rs2
alu_srcA
UType
IType
SType
JType
BType
PC
Immed
Gen
alu_srcB
Target
Gen
Branch
CondGen
pcSource
ALU
alu_fun
IOBUS_ADDR
memWrite
memRead2
ir[14:12]
size
Mem
adr2
din2
dout2
rs2
IOBUS_OUT
IOBUS_IN
IOBUS_WR
memBusy2
CSR reg
PC + 4
rf_wr_sel
Reg
File
wd
ir[11:7]
regWrite
wa
en
OTTER
Architecture
V3.0

Arm v2

- Introduced in 1986
- 58 instructions 24 mnemonics
- Added multiply and coprocessor instructions
- Coprocessor instructions to move, load, store data when working with coprocessors

Arm 610

- Apple Newton MessagePad 100 (1993)
- Apple invested in Acorn Computers who developed a specific ARM6-based RISC processor for the device



- **High performance RISC**
  15 MIPS sustained @ 20 MHz (20 MIPS peak)
- **Memory Management Unit** (MMU) support for virtual memory systems
- **4 kByte of instruction & data cache**
- **Write Buffer - enhancing performance**
- **Fully static operation - low power consumption**
  ideal for power sensitive applications
- **Fast sub microsecond interrupt response** for real-time applications
- **Excellent high-level language support**
- **Big and Little Endian operating modes**
- IEEE 1149.1 **Boundary Scan**
- **144 Thin Quad Flat Pack (TQFP) package**

Arm v3

- Introduced in 1991
- 32-bit address space, allowing direct access to up to 4 GB of memory.
- ARMv3 introduced support for virtual memory, enabling compatibility with modern operating systems that rely on memory management units (MMUs).
- ARMv3 included a compatibility mode to support the 26-bit addresses of earlier versions of the architecture. This **compatibility mode optional in ARMv4, and removed entirely in ARMv5.**
- Issue instructions to coprocessors without waiting on there execution enabling parallel processing
- Approximately 70 instructions

Arm v4

- Introduced in 1998
- **Thumb (ARMv4T only)** – A 16-bit compressed instruction set that improved code density for embedded systems. Only available in **ARMv4T**, not in standard ARMv4.
- Halfword and Signed Data Load/Store Operations: ARMv4 added support for loading and storing halfword data types, as well as signed byte and halfword data, which improved the handling of smaller data types in memory operations.
- New Privilege Mode: It introduced an additional kernel-level privilege mode, enhancing the architecture's capability to handle operating system-level tasks more efficiently.
- Enhanced Instruction Set: The instruction set was expanded with features like folding shifts and rotates into arithmetic and logical operations, allowing for more compact and efficient code execution
- ARMv4 included a general coprocessor interface that allowed **external FPUs (FPV1)**
- **ARMv4 (without Thumb):** ~90–100 instructions
- **ARMv4T (with Thumb):** ~120–140 total (counting both ARM & Thumb)

Arm Thumb Extension - 1997

- Background:
  - Arm supports “predication”, in which many instructions reserve opcode bits to function as a built in conditional
  - Eliminating the need for explicit branch instructions that depend on predicting which path will be taken. This reduces the penalties related to branch misprediction, such as pipeline flushes and stalls, and allows smoother instruction flow.
- Thumb mode
  - Processor can switch to 16 bit instruction mode, allowing 2x the instructions to fit in the 32 bit instruction cache
  - Tradeoff is: no predication, only explicit branch instructions
- Thumb 2 released in 2003
  - Introduced some 32 bit instructions to Thumb mode

https://developer.arm.com/documentation/dui0473/m/overview-of-the-arm-architecture/arm--thumb--and-thumbee-instruction-sets

Arm v5

- Introduced in 1998
- Thumb-2 Instruction Set: ARMv5 extended support for the Thumb instruction set, which allows for higher code density by using 16-bit instructions alongside 32-bit instructions. This improvement made ARMv5 more efficient for embedded systems with limited memory .
- DSP Enhancements: ARMv5TE introduced new Digital Signal Processing (DSP) instructions, such as signed multiply-accumulate and saturated arithmetic operations.
- FPv2 was introduced as an optional extension in ARMv5TE and ARMv5TEJ. This FPU supported IEEE 754-compliant single- and double-precision floating-point operations,
- Improved Exception Handling
- Enhanced Multiply Instructions: New multiply instructions were introduced, including support multiply-accumulate, which reduced the number of cycles required for such computations
- **Total (ARM + Thumb):** ~150–160 instructions
- **With Optional VFP (Floating Point):** ~180+

Multiply and Accumulate (MAC HArdware)

The multiply-accumulate operation involves two main steps:

1. **Multiply**: Compute the product of two numbers (operands).
2. **Accumulate**: Add the product to a running total (accumulator).

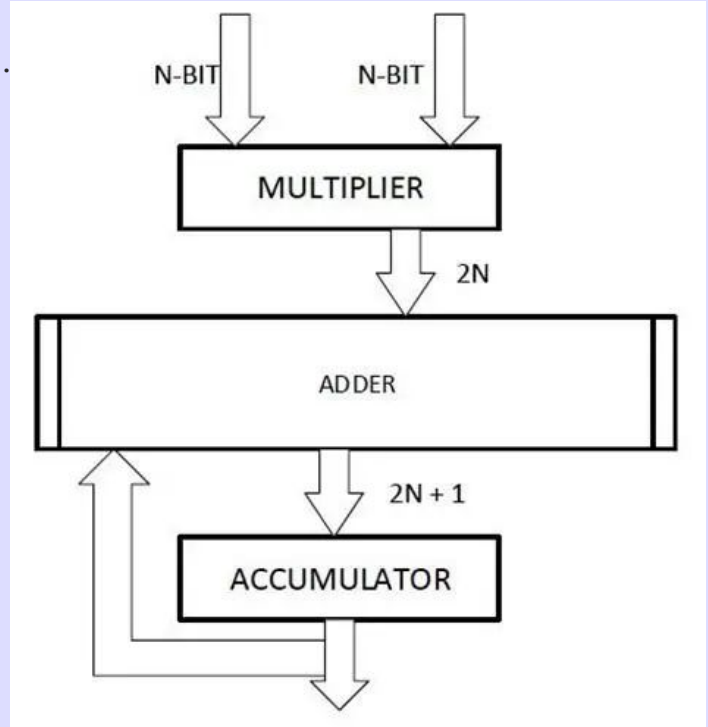
This operation can be mathematically represented as:

$$accumulator = accumulator + (A \times B)$$

$$accumulator = accumulator + (A \times B)$$

MLA R0, R1, R2, R3 ⇔

```
MUL R0, R1, R2
ADD R0, R0, R3
```

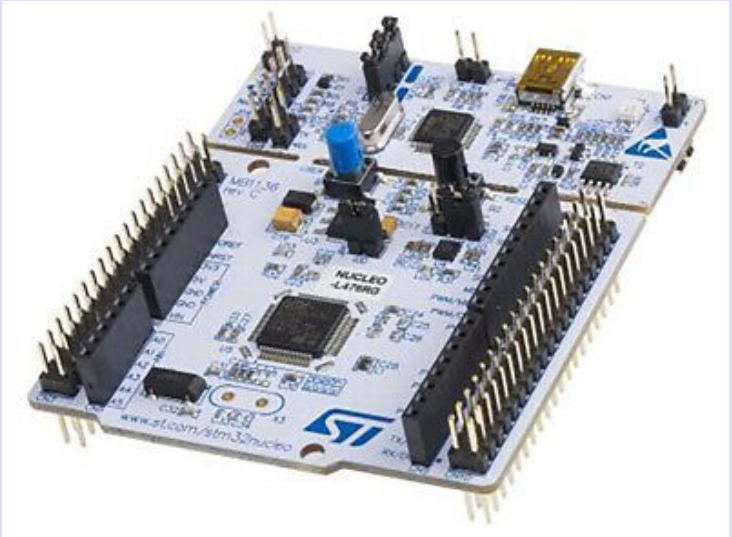


Armv6

- SIMD Extensions: Boosted multimedia performance for audio, video, and 3D graphics.
- Better Memory Handling: Allowed unaligned data access, mixed-endian systems, and improved caches for faster performance.
- Multiprocessing Support: Added instructions for synchronization and shared memory in multiprocessor systems (e.g., LDREX/STREX).
- Debugging Features: Introduced a formal debug architecture for easier external debugging.
- Multimedia Optimization: Improved video encoding and motion estimation for better media processing
- **ARM mode:** ~130–150
- **Thumb mode:** ~40–50
- **With SIMD & DSP extensions:** ~170–200
- **With VFP (Optional Floating Point Unit):** ~200–250

Armv7

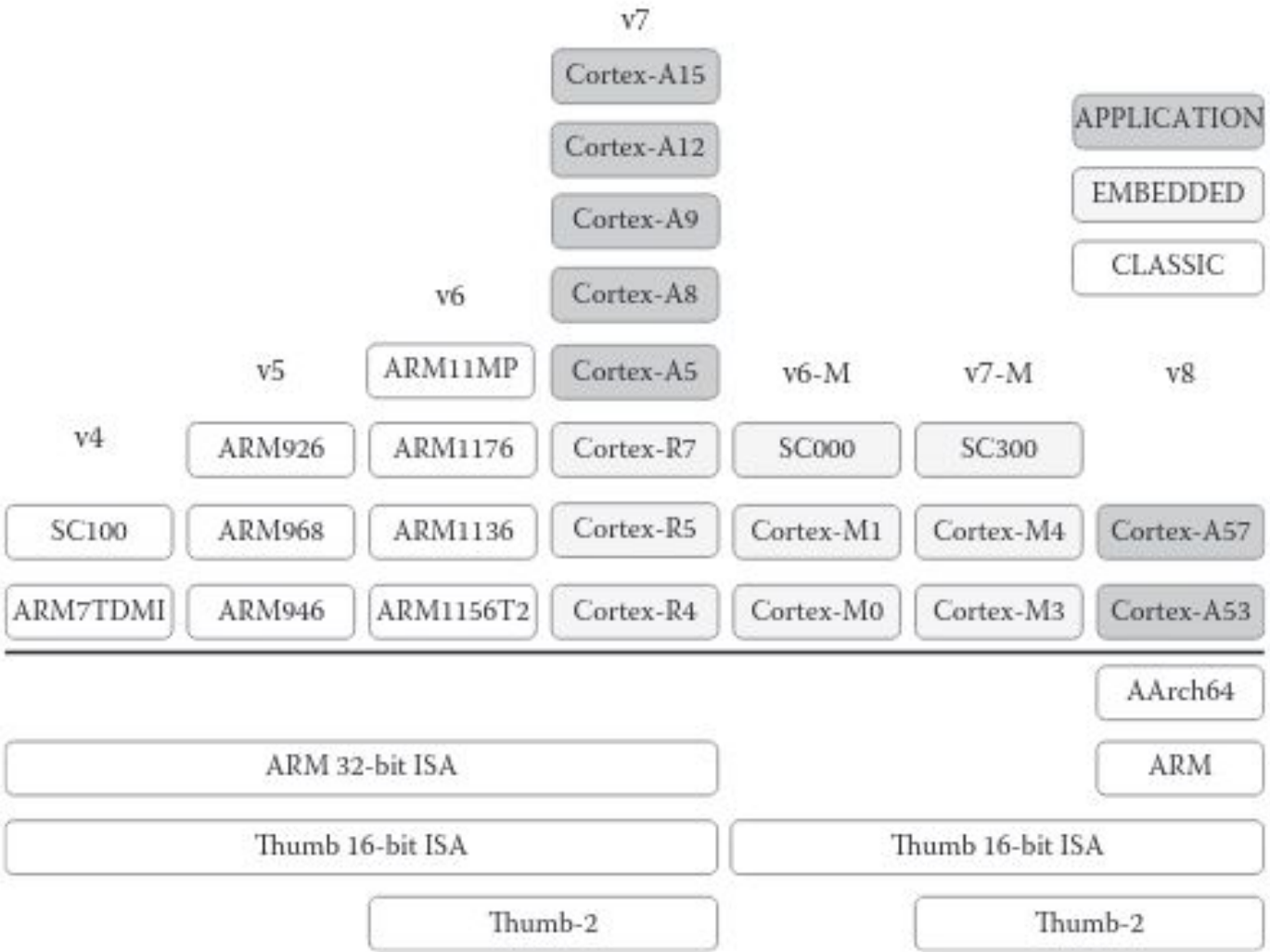
- Introduced in 2005
- Profiles for Different Applications: A,R,M
- Thumb-2 Technology - combed 16-bit and 32-bit instructions, reducing memory usage by up to 31% while improving performance by up to 38%.
- NEON SIMD Extensions
- TrustZone Security Expansion
- Advanced Memory Management: Added features like the Large Physical Address Extension (LPAE) for addressing up to 40-bit physical memory and improved cache policies for better performance.
- **Total (ARM + Thumb + Thumb-2, without optional VFP and NEON):** ~200–250
- **With Optional VFP & NEON**: ~300–350



Cortex M4


Cortex-A9
Cortex-A15
ARMv7-A
High Performance
32-bit CPU with
enterprise class
feature set
Cortex-A17
ARMv7-A
High Performance
32-bit CPU with
lower power and
smaller area
Cortex-A57
ARMv8-A
Highest
performance
64/32-bit CPU
High
Performance
A
Cortex-A5
ARMv7-A
Smallest and
lowest power CPU
Cortex-A7
ARMv7-A
High efficiency
32-bit CPU
big LITTLETM
compatible
Cortex-A53
ARMv8-A
High efficiency
64/32-bit CPU
big LITTLETM
compatible
High
Efficiency
R
Cortex-R4
Real-time standard
Cortex-R5
Functional safety
Cortex-R7
High performance
4G modern and
storage
Real-time
Enhanced system integration features
M
Cortex-M0
Lowest cost
Lowest power
Cortex-M0+
Highest energy
efficiency
Cortex-M3
Performance
efficiency
Cortex-M4
Mainstream
Control & DSP
Cortex-M7
Maximum
Performance
Control & DSP
Control

	Feature	Cortex-M4	Cortex-A	Cortex-R
2	Application Focus	Microcontroller, embedded control	High-performance application cores	Real-time, safety-critical systems
3	Pipeline Complexity	Simple 3-stage, in-order	Complex, superscalar, out-of-order	Medium complexity, dual/triple-issue
4	Clock Speed	Up to ~300 MHz	1 GHz to 3+ GHz	Up to 600 MHz and beyond
5	MMU Support	No	Full MMU for virtual memory	Optional MMU, Memory Protection Unit
6	OS Support	Real-time OS, bare-metal	Full OS (Linux, Android, Windows)	RTOS and safety-critical OS
7	Virtualization Support	No	Yes	Limited or no
8	Floating-Point Unit (FPU)	Optional single-precision	Single and double-precision FPU	Optional
9	DSP Extensions	Yes	Advanced SIMD & NEON	DSP instructions, error correction
10	Error Correction & Fault Tol	Basic error handling	Not typically prioritized	Lockstep cores, ECC, fault detection
11	Power Consumption	Low	Higher (performance tradeoff)	Moderate (performance & reliability tradeoff)
12	Interrupt Latency	Low, deterministic	Higher	Very low, deterministic
13	Target Use Cases	Low power embedded control	Smartphones, tablets, networking	Automotive, industrial, medical devices
14				


v7
Cortex-A15
Cortex-A12
Cortex-A9
Cortex-A8
Cortex-A5
Cortex-R7
Cortex-R5
Cortex-R4
v6
ARM11MP
ARM1176
ARM1136
ARM1156T2
v5
ARM926
ARM968
ARM946
v4
SC100
ARM7TDMI
v6-M
SC000
Cortex-M1
Cortex-M0
v7-M
SC300
Cortex-M4
Cortex-M3
v8
Cortex-A57
Cortex-A53
APPLICATION
EMBEDDED
CLASSIC
AArch64
ARM
ARM 32-bit ISA
Thumb 16-bit ISA
Thumb-2
Thumb 16-bit ISA
Thumb-2

Arm v8

- 64-bit Architecture (AArch64): ARMv8 introduced a 64-bit execution mode, enabling larger memory addressing and improved performance for high-end applications while retaining backward compatibility with 32-bit (AArch32) code.
  Doubling the instructions.
- Improved Performance: Enhanced instruction sets, including support for SIMD and cryptographic operations, boosted performance for multimedia and security applications.
- Virtualization Support: Added hardware-assisted virtualization, allowing multiple operating systems to run efficiently on the same hardware.
- TrustZone Security: Extended TrustZone technology to ARMv8-M for embedded systems.
- Enhanced Debugging and Memory Management.
- A64 base instruction set contains 354 instructions, with additional sets for SIMD (404) and SVE (508) instructions

Arm v9

- Introduced in 2021
- SVE2: Improved vector processing for AI, ML, and DSP.
- Security: Added Realm Management (RME) for confidential computing and Memory Tagging (MTE) for memory safety.
- Performance: Promised up to 30% better performance over ARMv8 with improved efficiency.
- Transactional Memory (TME): Enhanced multi-threaded performance.
- System-Level Optimizations: Focused on better cache and memory management.

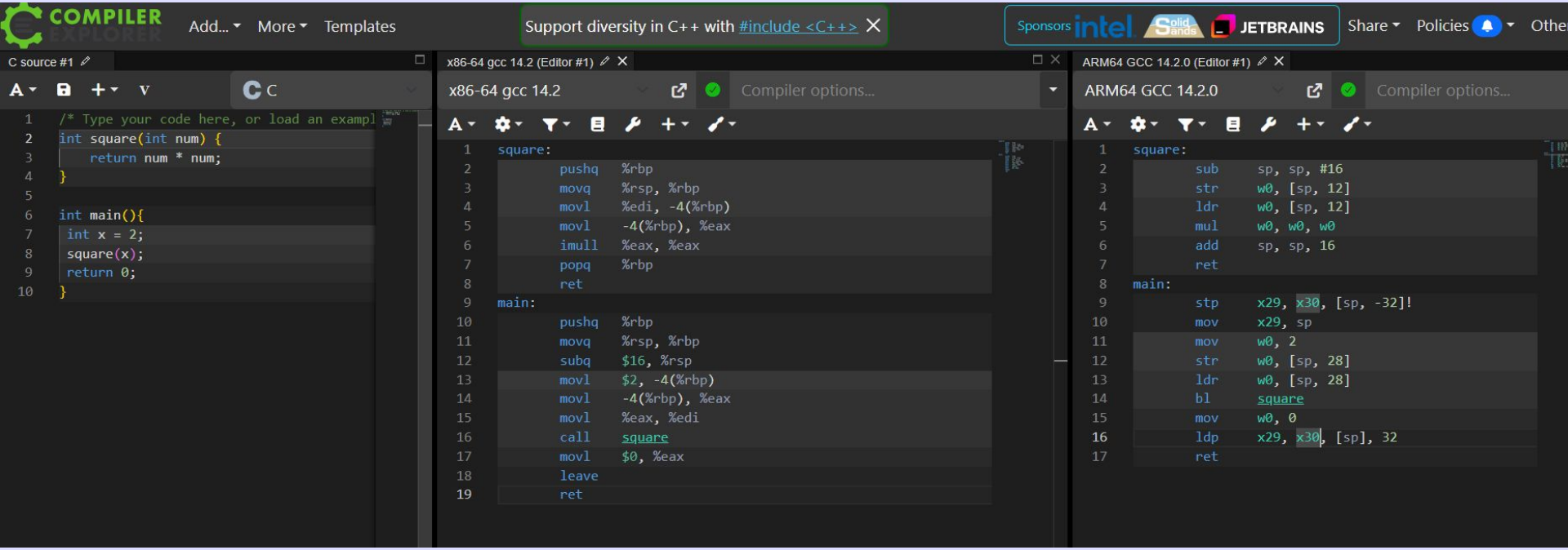
IS ARM CISC or RISC?

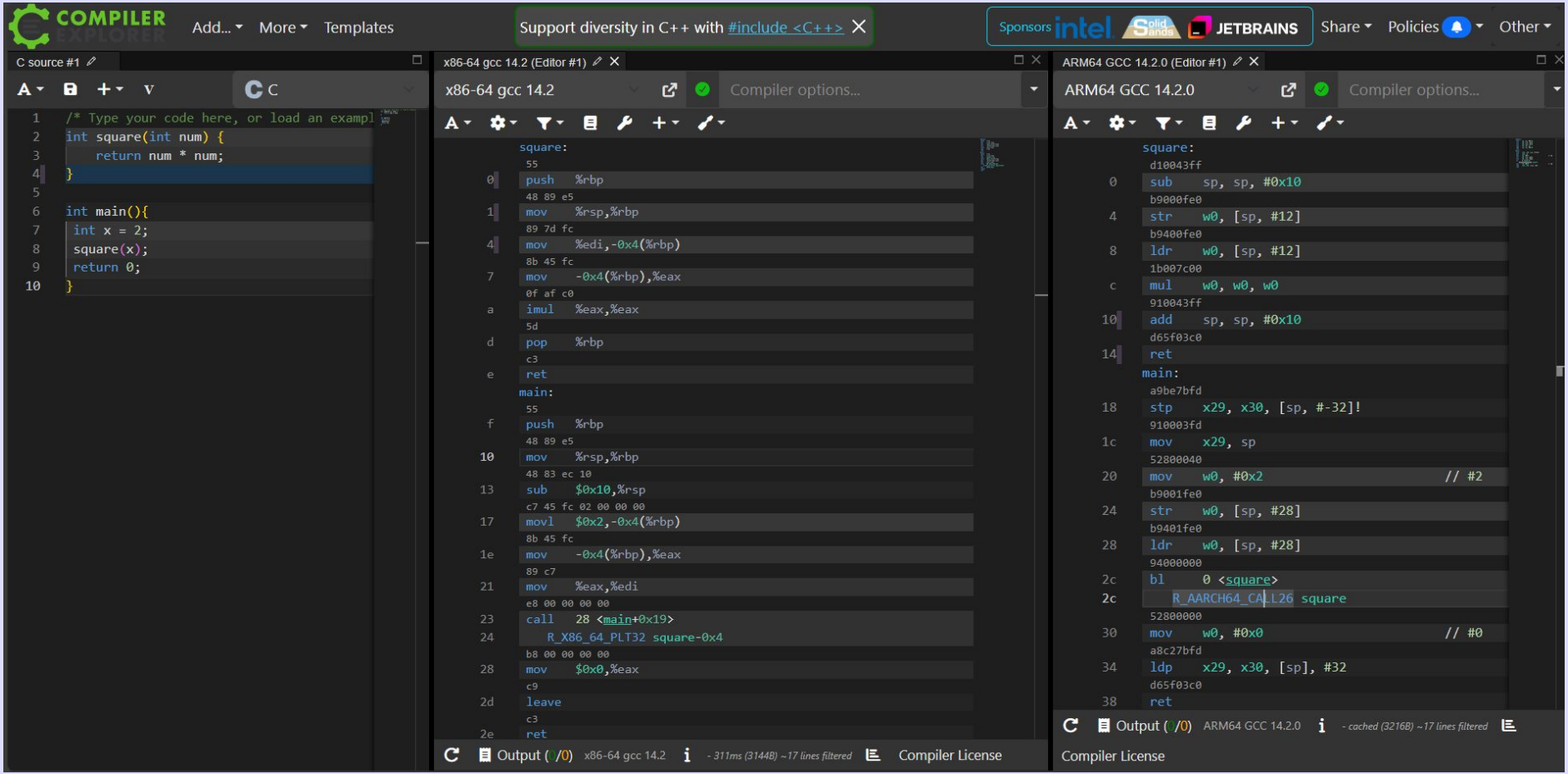
https://chipsandcheese.com/p/arms-cortex-a53-tiny-but-important
https://soclabs.org/technology/cortex-a53

If you include every extension in an ARMv9 the total number of instructions would exceed 1,500.

If you only count the operators in the reference materials, there are over **1500 unique instructions in Modern X86** [1]

What's Really The Difference between ARM and Intel ISA?


COMPILER EXPLORER
Add...
More
Templates
Support diversity in C++ with #include <C++>
Sponsors intel Solid Sands JETBRAINS
Share
Policies
C source #1
C
/* Type your code here, or load an exampl
int square(int num) {
return num * num;
}
int main(){
int x = 2;
square(x);
return 0;
}
x86-64 gcc 14.2 (Editor #1)
x86-64 gcc 14.2
Compiler options...
square:
pushq %rbp
movq %rsp, %rbp
movl %edi, -4(%rbp)
movl -4(%rbp), %eax
imull %eax, %eax
popq %rbp
ret
main:
pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl $2, -4(%rbp)
movl -4(%rbp), %eax
movl %eax, %edi
call square
movl $0, %eax
leave
ret
ARM64 GCC 14.2.0 (Editor #1)
ARM64 GCC 14.2.0
Compiler options...
square:
sub sp, sp, #16
str w0, [sp, 12]
ldr w0, [sp, 12]
mul w0, w0, w0
add sp, sp, 16
ret
main:
stp x29, x30, [sp, -32]!
mov x29, sp
mov w0, 2
str w0, [sp, 28]
ldr w0, [sp, 28]
bl square
mov w0, 0
ldp x29, x30, [sp], 32
ret


COMPILER EXPLORER
Add...
More
Templates
Support diversity in C++ with #include <C++>
Sponsors intel Solid Sands JETBRAINS
Share
Policies
Other
C source #1
C
/* Type your code here, or load an exampl
int square(int num) {
return num * num;
}
int main(){
int x = 2;
square(x);
return 0;
}
x86-64 gcc 14.2 (Editor #1)
x86-64 gcc 14.2
Compiler options...
square:
55
push %rbp
48 89 e5
mov %rsp,%rbp
89 7d fc
mov %edi,-0x4(%rbp)
8b 45 fc
mov -0x4(%rbp),%eax
0f af c0
imul %eax,%eax
5d
pop %rbp
c3
ret
main:
55
push %rbp
48 89 e5
mov %rsp,%rbp
48 83 ec 10
sub $0x10,%rsp
c7 45 fc 02 00 00 00
movl $0x2,-0x4(%rbp)
8b 45 fc
mov -0x4(%rbp),%eax
89 c7
mov %eax,%edi
e8 00 00 00 00
call 28 <main+0x19>
R_X86_64_PLT32 square-0x4
b8 00 00 00 00
mov $0x0,%eax
c9
leave
c3
ret
Output (/0) x86-64 gcc 14.2 - 311ms (3144B) ~17 lines filtered Compiler License
ARM64 GCC 14.2.0 (Editor #1)
ARM64 GCC 14.2.0
Compiler options...
square:
d10043ff
sub sp, sp, #0x10
b9000fe0
str w0, [sp, #12]
b9400fe0
ldr w0, [sp, #12]
1b007c00
mul w0, w0, w0
910043ff
add sp, sp, #0x10
d65f03c0
ret
main:
a9be7bfd
stp x29, x30, [sp, #-32]!
910003fd
mov x29, sp
52800040
mov w0, #0x2 // #2
b9001fe0
str w0, [sp, #28]
b9401fe0
ldr w0, [sp, #28]
94000000
bl 0 <square>
R_AARCH64_CALL26 square
52800000
mov w0, #0x0 // #0
a8c27bfd
ldp x29, x30, [sp], #32
d65f03c0
ret
Output (/0) ARM64 GCC 14.2.0 - cached (3216B) ~17 lines filtered
Compiler License

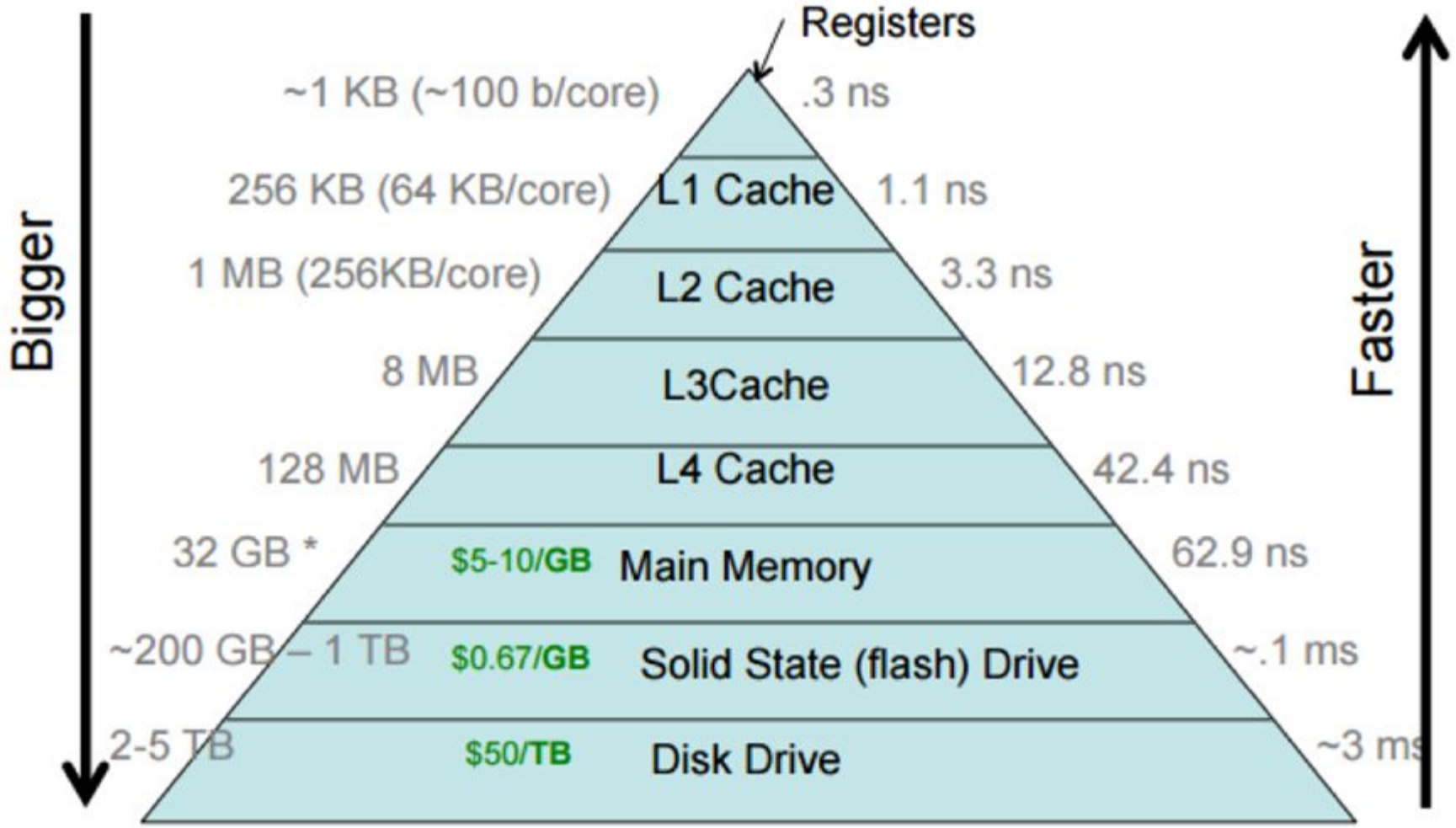
How much space Does the Function Take Up>

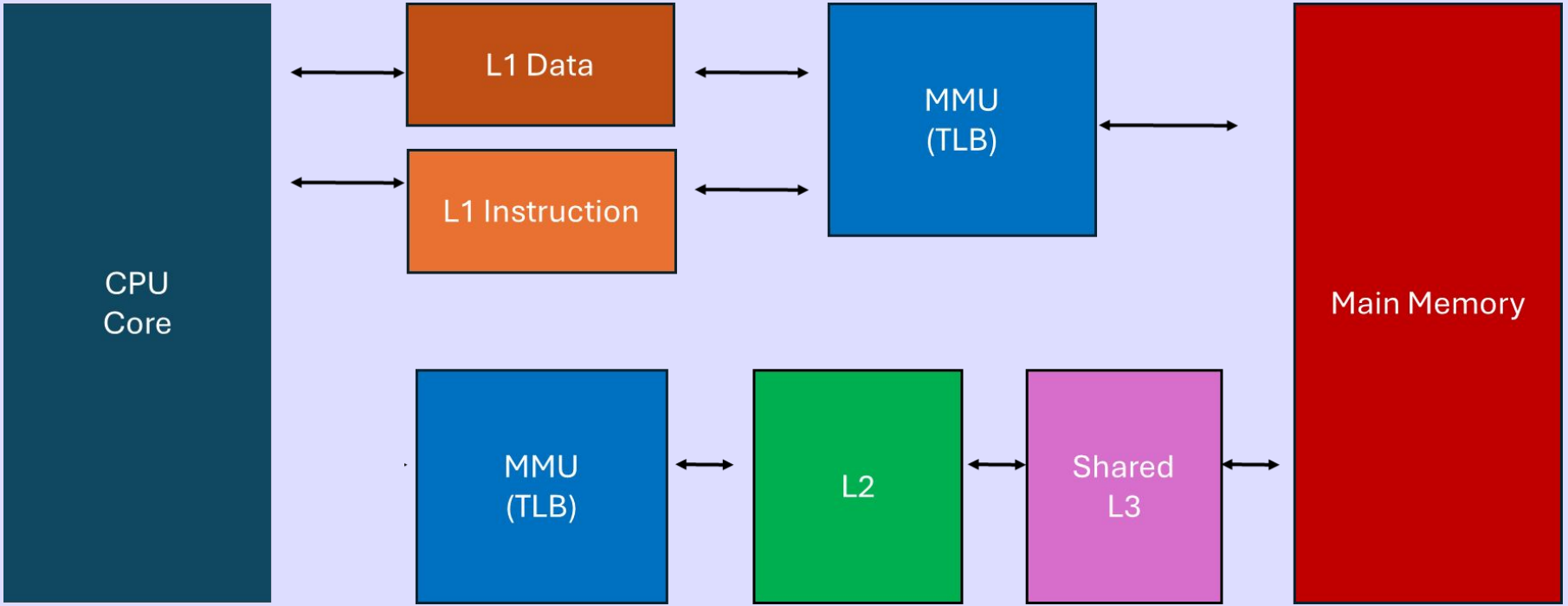
```c
/* Type your code here, or load an example. */
int square(int num) {
    return num * num;
}

int main(){
 int x = 2;
 square(x);
 return 0;
}
```

ARM -> 56 Bytes

x86 -> 47 Bytes


Registers
~1 KB (~100 b/core)
.3 ns
256 KB (64 KB/core)
L1 Cache
1.1 ns
1 MB (256KB/core)
L2 Cache
3.3 ns
8 MB
L3Cache
12.8 ns
128 MB
L4 Cache
42.4 ns
32 GB *
$5-10/GB
Main Memory
62.9 ns
~200 GB – 1 TB
$0.67/GB
Solid State (flash) Drive
~.1 ms
2-5 TB
$50/TB
Disk Drive
~3 ms
Bigger
Faster


CPU
Core
L1 Data
L1 Instruction
MMU
(TLB)
MMU
(TLB)
L2
Shared
L3
Main Memory

```
[av@think-av ~]$ lscpu --cache
NAME ONE-SIZE ALL-SIZE WAYS TYPE        LEVEL SETS PHY-LINE COHERENCY-SIZE
L1d       32K      64K    8 Data            1   64        1             64
L1i       32K      64K    8 Instruction     1   64        1             64
L2       256K     512K    8 Unified         2  512        1             64
L3         3M       3M   12 Unified         3 4096        1             64
```

X86

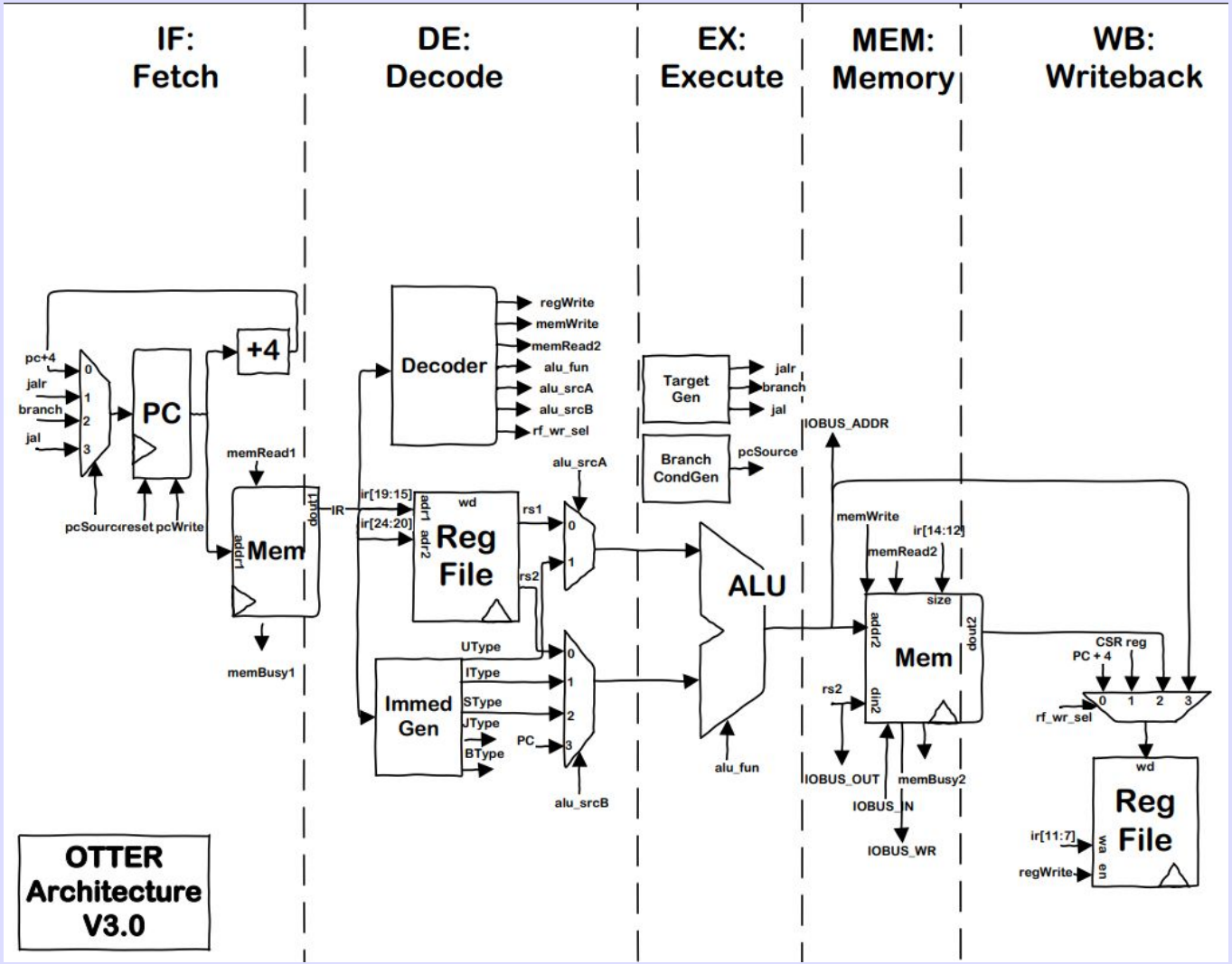
More Instruction Cache Hits

Variable Instructions -> Complex Slower Decodder

ARM64

More Cache Misses

Fixed Instructions -> means faster / more decoders


IF:
Fetch
DE:
Decode
EX:
Execute
MEM:
Memory
WB:
Writeback
pc+4
jalr
branch
jal
PC
+4
pcSource
reset
pcWrite
memRead1
Mem
addr1
dout1
IR
memBusy1
Decoder
regWrite
memWrite
memRead2
alu_fun
alu_srcA
alu_srcB
rf_wr_sel
ir[19:15]
ir[24:20]
adr1
adr2
wd
Reg
File
rs1
rs2
UType
IType
SType
JType
BType
Immed
Gen
PC
alu_srcA
alu_srcB
Target
Gen
Branch
CondGen
jalr
branch
jal
pcSource
ALU
alu_fun
IOBUS_ADDR
memWrite
memRead2
ir[14:12]
size
addr2
din2
dout2
Mem
rs2
IOBUS_OUT
IOBUS_IN
IOBUS_WR
memBusy2
CSR reg
PC + 4
rf_wr_sel
wd
Reg
File
ir[11:7]
regWrite
wa
en
OTTER
Architecture
V3.0

RISCV Extensions To Run Modern 64bit linux

- RV64I (Base Integer): 47 unique instructions.
- M (Multiply/Divide): 8 instructions.
- A (Atomic): 12 instructions.
- F (Single-Precision Floating Point): ~34 instructions.
- D (Double-Precision Floating Point): ~39 instructions.
- C (Compressed): ~40 instructions (compressed instructions count separately).
- Zicsr (Control/Status Registers): ~6 instructions.
- Zifencei (Instruction Fence): 1 instruction.
- Privileged Instructions (MMU + Supervisor mode): ~20–25 unique instructions (syscalls, paging, traps, etc.).

RISCV ISA vs ARM

ISA	**Unique Mnemonics**	**Including Operand Variants (Approx.)**	**Notes**
RISC-V RV64GC + MMU	~212	~400-500	Simpler modular variations, limited operand variants
ARMv8-A (AArch64)	~442	1300-2000+	Large operand variants, many SIMD/vector options

The Other Real Difference Business Stuff


Cortex-A9
Cortex-A15
ARMv7-A
High Performance
32-bit CPU with
enterprise class
feature set
Cortex-A17
ARMv7-A
High Performance
32-bit CPU with
lower power and
smaller area
Cortex-A57
ARMv8-A
Highest
performance
64/32-bit CPU
High
Performance
A
Cortex-A5
ARMv7-A
Smallest and
lowest power CPU
Cortex-A7
ARMv7-A
High efficiency
32-bit CPU
big LITTLETM
compatible
Cortex-A53
ARMv8-A
High efficiency
64/32-bit CPU
big LITTLETM
compatible
High
Efficiency
R
Cortex-R4
Real-time standard
Cortex-R5
Functional safety
Cortex-R7
High performance
4G modern and
storage
Real-time
Enhanced system integration features
M
Cortex-M0
Lowest cost
Lowest power
Cortex-M0+
Highest energy
efficiency
Cortex-M3
Performance
efficiency
Cortex-M4
Mainstream
Control & DSP
Cortex-M7
Maximum
Performance
Control & DSP
Control

Unprecedented Access to Industry-leading IP

Processor IP

CPUs

- Cortex-A53
- Cortex-A35
- Cortex-A34
- Cortex-A32
- Cortex-A7
- Cortex-A5
- Cortex-R52
- Cortex-R8
- Cortex-R5
- Cortex-M33
- Cortex-M23
- Cortex-M7
- Cortex-M4
- Cortex-M3
- Cortex-M0+
- Cortex-M0

75% of Arm Cortex licenses over past 2 years

GPUs

- Mali-G52
- Mali-G31

Includes Mali drivers (DDK)

Interconnect

- CoreLink NIC-450
- CoreLink NIC-400
- CoreLink CCI-400
- CoreLink CCI-500
- CoreLink CCI-550
- ADB-400 AMBA domain bridge
- XHB-400 AXI-AHB bridge

System Controllers

- CoreLink GIC-500
- CoreLink GIC-400
- PL192 VIC
- BP141 TrustZone Mem. Wrapper
- CoreLink TZC-400
- CoreLink L2C-310
- CoreLink MMU-500
- BP140 Mem. Intf.

Security IP

- CryptoCell-312
- CryptoCell-712
- TrustZone True Random Number Generator

Peripherals

- PL011 UART
- PL022 SPI
- PL031 RTC

Debug & Trace

- CoreSight SoC-400
- CoreSight SDC-600
- CoreSight STM-500
- CoreSight System Trace Macrocell
- CoreSight Trace Memory Controller

Design Kits

- Corstone-101
- Corstone-201

Physical IP

- Artisan physical IP platform, TSMC
- Artisan PIK for Cortex-M33, TSMC 22ULL

Tools & Models

- Socrates IP Tooling
- Arm Design Studio
- Virtual system models (fast & cycle accurate)

Support

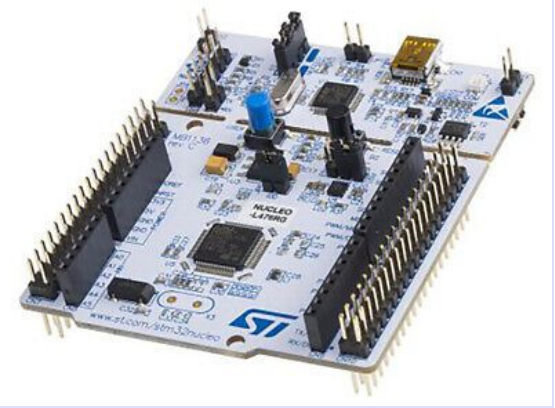
- Standard Arm technical support
- Arm online training
- Maintenance updates
- Credits towards onsite training and design reviews

UNDER EMBARGO UNTIL 6:00 AM PDT 16th July 2019

arm

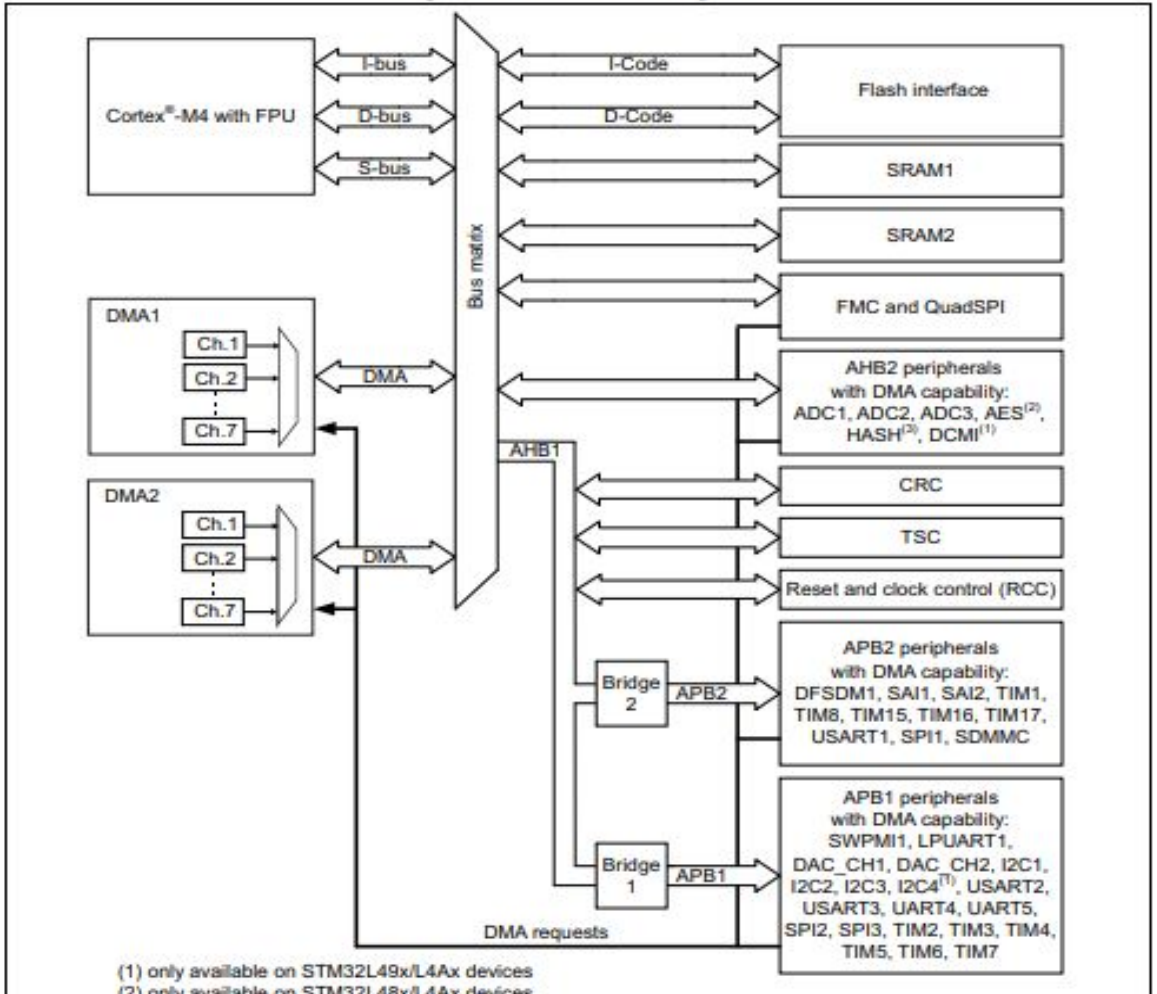
	 **Cortex-M0** Exceptionally small silicon area Download Now Please register for or login to an Arm Account to download the IP	 **Cortex-M3** For highly deterministic apps Download Now Please register for or login to an Arm Account to download the IP
Includes	• Cortex-M0 • Cortex-M System Design Kit (CMSDK)	• Cortex-M3 • Corstone-101 reference package which includes SSE-050 subsystem (pre-validated and assembled) and several IP blocks for added security
Tools and Models	• 90-day trial of Arm Keil MDK • 90-day trial of IAR Embedded Workbench • FPGA prototyping board for Cortex-M	• 90-day trial of Arm Keil MDK • 90-day trial of IAR Embedded Workbench • FPGA prototyping board for Cortex-M • 1-year access to Cortex-M3 cycle model

AMBA Bus for DMA



The DMA block diagram is shown in the figure below.

Figure 31. DMA block diagram



(1) only available on STM32L49x/L4Ax devices

RISC-V-SoC-Design

Public

Watch 1 Fork 0 Star 13

main 1 Branch 0 Tags Go to file t Add file Code

nietzhuang Update PC.sv 842f476 · 4 years ago 92 Commits

TOP	Update PC.sv	4 years ago
README.md	Update README.md	4 years ago

README

RISC-V-SoC-Design

Single RISC-V CPU attached on AMBA AHB with Instruction and Data memories.

Use ncverilog

- ncverilog top_tb.sv +define+progX +access+r +sv +notimingcheck (X is 0 to 4)

About

Single RISC-V CPU attached on AMBA AHB with Instruction and Data memories.

Readme
Activity
13 stars
1 watching
0 forks
Report repository

Releases

No releases published

Packages

No packages published

Languages

SystemVerilog 64.3% Verilog 34.0% Assembly 1.7%

Apple Silicon (P/E Cores)

- Apple has an “Architecture License”, meaning they design their own Arm compatible core design
  - After starting with mobile arm processors, they transitioned to desktop class

- A reasonable middle ground for consumers
- Extra effort in terms of compatibility
  - Rosetta
- https://en.wikipedia.org/wiki/Apple_silicon

Is RISC and CISC Still an Important Distinction In Modern Architecture?
NO

END

Extra Content

Flexibility (examples and extensions)

SSE (Intel x86) vs. NEON (ARM)

- Extensions of SIMD to compute large amount of a data in parallel, for the GPU poor
  - Store vectors in registers and operate on vectors simultaneously
  - Multi-cycle instructions -> One instruction

- New Registers:
  - **Sixteen new 128-bit registers** (XMM0–XMM15)
    - eXtended MultiMedia
- Data Types Supported in Registers:
  - Packed single-precision **(32-bit) floats**
    - Eg: { f1, f2, f3, f4 }
- Instruction Categories:
  - Arithmetic, Data Movement, Bitwise Operations, Data Shuffling

- New Registers:
  - **Sixteen new 128-bit registers** (Q0–Q15)
    - Quadword (register width)
    - Can be accessed as 32 64-bit (D0–D31) or 64 32-bit (S0–S63)
- Data Types Supported in Registers:
  - Packed single-precision **(32-bit) floats**
  - Packed **8-bit, 16-bit, 32-bit, and 64-bit integers.**
- Instruction Categories:
  - Arithmetic, Data Movement, Bitwise Operations, Data Shuffling

SSE (Intel x86) vs. NEON (ARM) - C Data Types

Read as

Sixteen new 128-bit registers

- **Multimedia**
- **128: Number of bits in the vector register**

```
__m128 // Represents a 128-bit vector for
single-precision floating-point values
__m128i // Represents a 128-bit vector for integer
values
__m128d // Represents a 128-bit vector for
double-precision floating-point values
```

```
typedef uint8x16_t  uint8x16_t;
typedef int8x16_t   int8x16_t;
typedef uint16x8_t uint16x8_t;
typedef int16x8_t int16x8_t;
typedef uint32x4_t uint32x4_t;
typedef int32x4_t   int32x4_t;   // 4 signed 32-bit
integers
typedef uint64x2_t  uint64x2_t;  // 2 unsigned
64-bit integers
typedef int64x2_t   int64x2_t;   // 2 signed 64-bit
integers
typedef float32x4_t float32x4_t; // 4 32-bit
floating point
typedef float64x2_t float64x2_t; // 2 64-bit
floating point
```

SSE (Intel x86) vs. NEON (ARM) - Vector Multiplication in C

- Extensions of SIMD to compute large amount of a data in parallel, for the GPU poor
  - Multi-cycle instructions -> One instruction

```c
#include <xmmintrin.h>

void vector_multiply_sse(float* a, float* b, float*
result, int size) {
    for (int i = 0; i < size; i += 4) {
        __m128 va = _mm_loadu_ps(a + i);
        __m128 vb = _mm_loadu_ps(b + i);
        __m128 vresult = _mm_mul_ps(va, vb);
        _mm_storeu_ps(result + i, vresult);
    }
}
```

```c
#include <arm_neon.h>

void vector_multiply_neon(float* a, float* b,
float* result, int size) {
    for (int i = 0; i < size; i += 4) {
        float32x4_t va = vld1q_f32(a + i);
        float32x4_t vb = vld1q_f32(b + i);
        float32x4_t vresult = vmulq_f32(va, vb);
        vst1q_f32(result + i, vresult);
    }
}
```

Arm Thumb Extension - 1997

- **Background:**
  - Arm supports “predication”, in which many instructions reserve opcode bits to function as a built in conditional
- **Thumb mode**
  - Processor can switch to 16 bit instruction mode, allowing 2x the instructions to fit in the 32 bit instruction cache
  - Tradeoff is: no predication, only explicit branch instructions
- **Thumb 2 released in 2003**
  - Introduced some 32 bit instructions to Thumb mode

https://developer.arm.com/documentation/dui0473/m/overview-of-the-arm-architecture/arm--thumb--and-thumbee-instruction-sets

Arm Jazelle - 2001

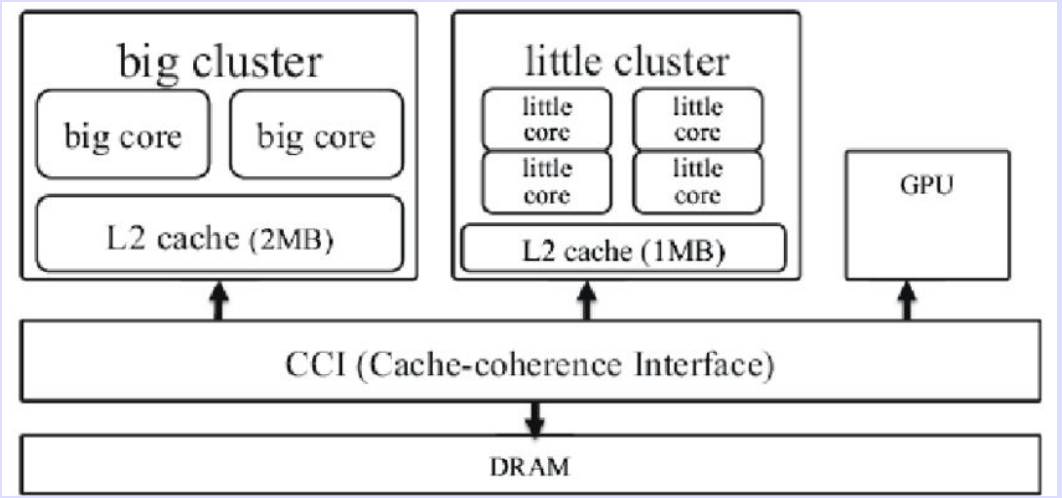
- **Jazelle - A Hardware Java Virtual Machine**
  - Specialized instructions that support Java bytecode
  - Stuff like hardware array bounds checking
  - Replaced by ThumbEE
- **ThumbEE (2005-2011)**
  - Optimized for Dynamically Generated code (interpreted or JIT compiled)
  - Mostly replaced by better compile/runtime optimizations, but the hardware still does a lot to make that possible

Core Designs

Arm big.LITTLE

- Straight from the horse's mouth:

  "big.LITTLE technology is a heterogeneous processing architecture that uses up to three types of processors. ***"LITTLE" processors are designed for maximum power efficiency****, while* ***"big" processors are designed to provide sustained compute performance****."*



How much processor do I need?

Throughput /
Multicore
Performance

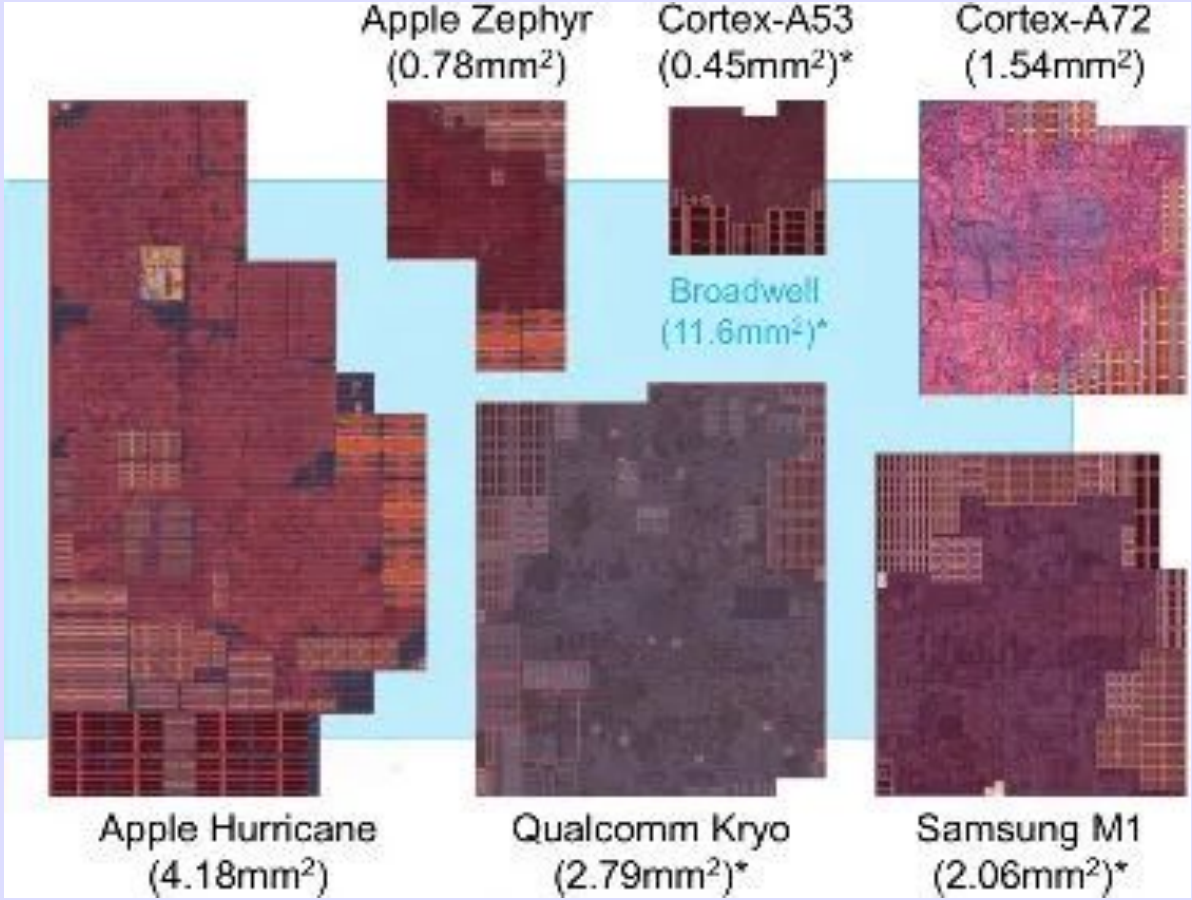
Latency /
Single Core
Performance

Compatibility

Power Efficiency

Die Shots of ARM Cores

- All are 64-bit ARMv8
  - Excluding the x86 Broadwell core in the back
- All are single cores
  - (no main memory, peripherals, accelerators, etc)
- And all are totally different.

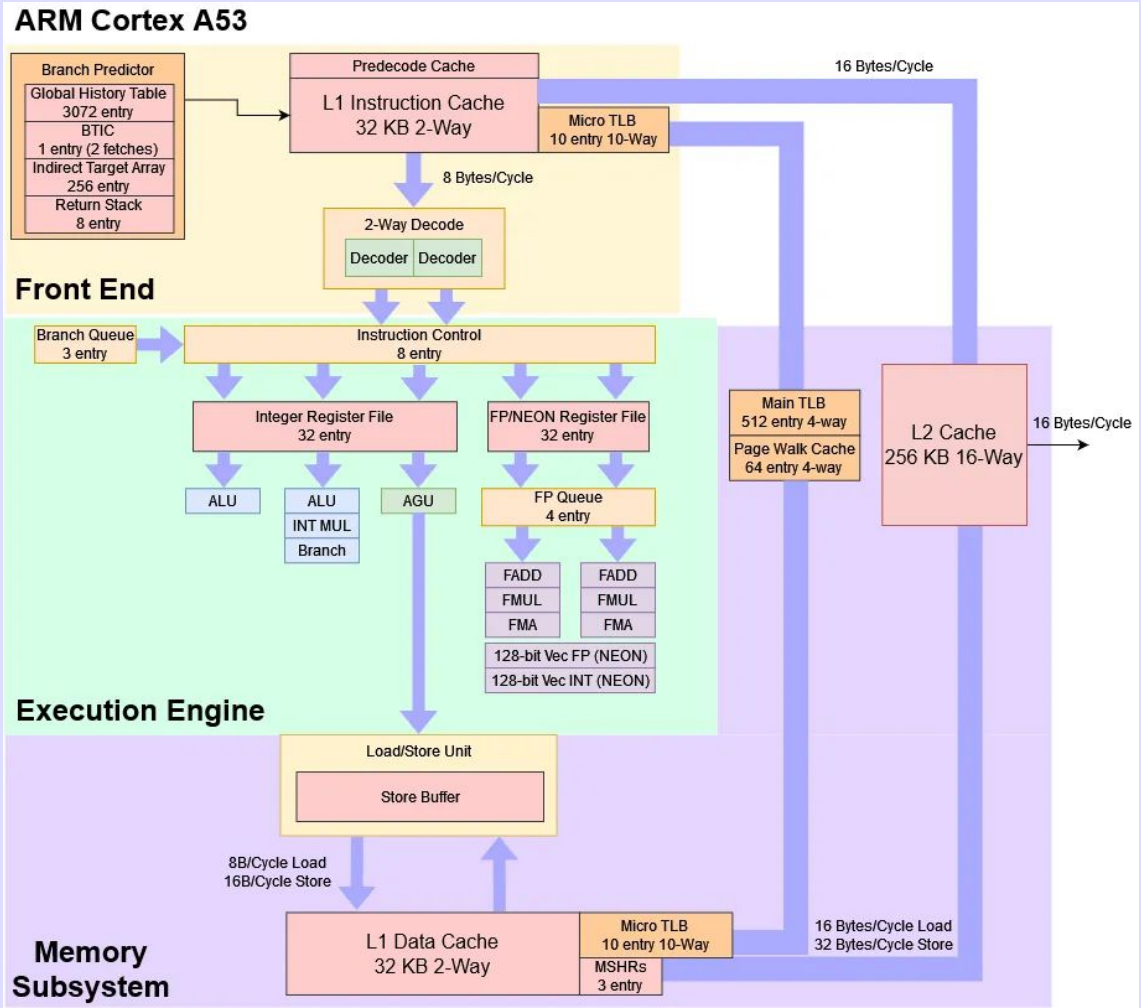


https://chipsandcheese.com/p/kryo-qualcomms-last-in-house-mobile-core

Cortex A53

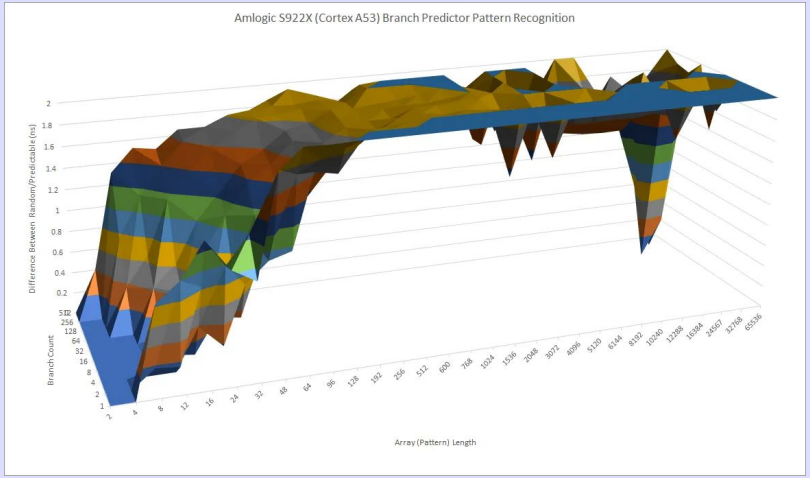
- 64-bit Armv8-A architecture
- one to four cores
- “automatic data cache coherency”
  - All parallel cores see same cached data, no matter how bad your program is
- (up to) 2MB L2 cache shared across cores
- Pipeline facts
  - 8-stages (fairly low #)
  - “symmetric dual-issue”
    - 2 instructions at once (plus the pipeline queueing)
  - “In-order”
    - Hardware **does not reorder or rearrange** queued instructions in the event of stalls or dependencies

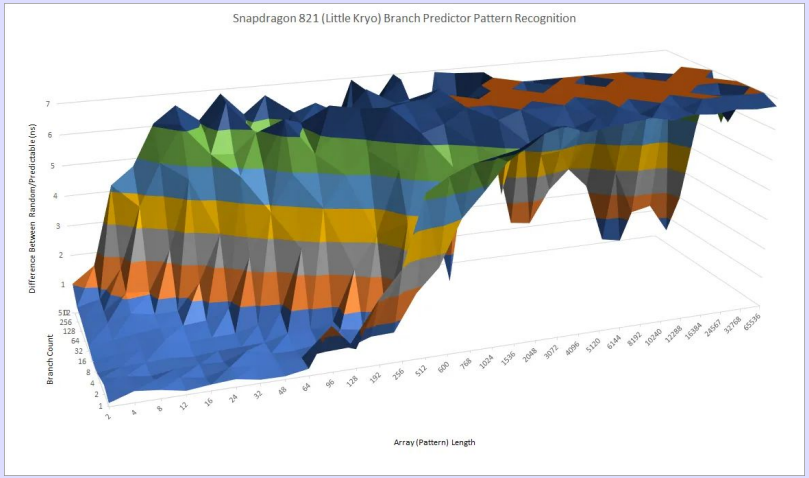
https://chipsandcheese.com/p/arms-cortex-a53-tiny-but-important
https://soclabs.org/technology/cortex-a53



https://chipsandcheese.com/p/arms-cortex-a53-tiny-but-important

Cortex A53 vs Snapdragon 821 (Little Kryo) Branch Predictor





https://chipsandcheese.com/p/arms-cortex-a53-tiny-but-important

Apple Silicon (P/E Cores)

- Apple has an “Architecture License”, meaning they design their own Arm compatible core design
  - After starting with mobile arm processors, they transitioned to desktop class

- A reasonable middle ground for consumers
- Extra effort in terms of compatibility
  - Rosetta
- https://en.wikipedia.org/wiki/Apple_silicon

Ampere Altra Max

- More powahh
- Architecture Design License
- 128-core Arm CPU, with support for up to 16 bit floating pt instructs

Samsung Exynos M1

- Good link, actual details on the architecture inside

- https://www.anandtech.com/show/10590/hot-chips-2016-exynos-m1-architecture-disclosed

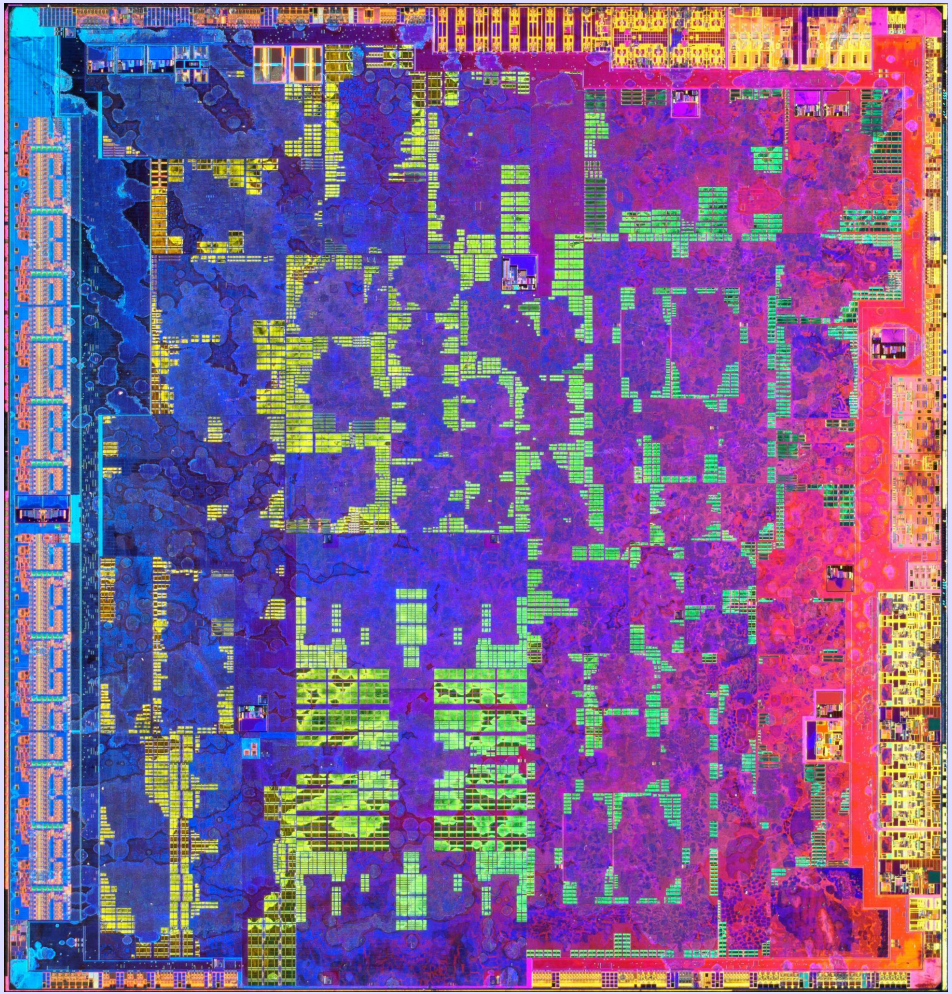
- Samsung cancelled their ARM core development in 2019, reopened in 2024

4B

Chips with Cores

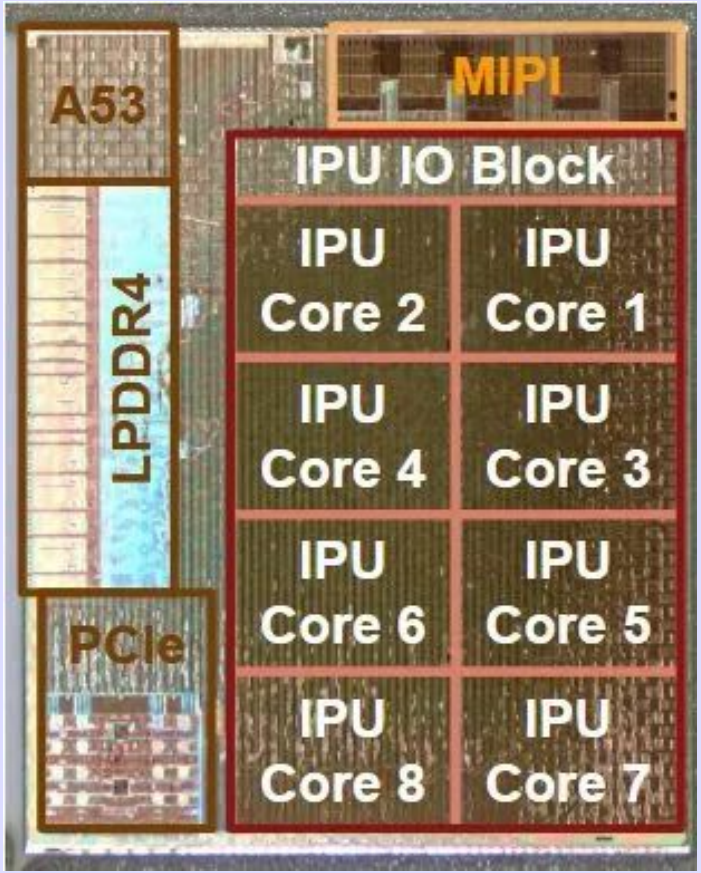
Nvidia Tegra X1 - 2015

- 4x ARM Cortex A53
  - Low power efficiency cores
- 4x ARM Cortex A57
  - High power **gaming** cores
- 2x Nvidia Maxwell SM GPUs
  - for **gaming**



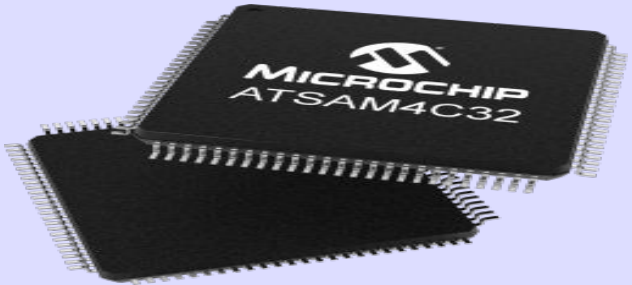
Google Pixel Visual Core - 2018

- 1x ARM Cortex A53
  - Manages the star of the show
- 8x Google IPU Cores
  - Is the star of the show
- MIPI Camera Interface
- LPDDR4 RAM
- PCIE
  - For getting the data somewhere useful



ATSAM4C32 Smart Meter Processor

- Landis+Gyr Residential Power Meter
- Uses ATSAM4C32 Processor
  - “Dual Arm® Cortex® -M4 Core SOC with Advanced Security Features for Residential and C&I Smart Meters”

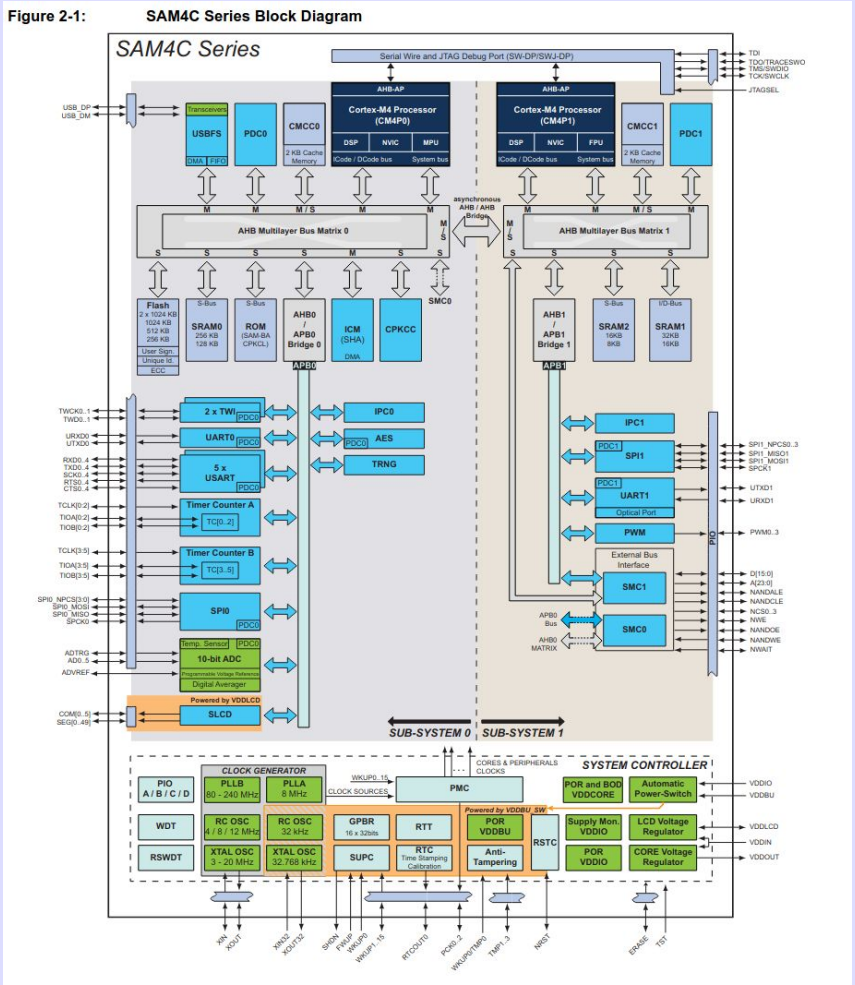


https://wiki.recessim.com/view/Advanced_Metering_Infrastructure

ATSAM4C32 Smart Meter Processor

- Coprocessor (provides ability to separate application, communication or metrology functions)
  - ARM Cortex-M4F running at up to 120 MHz(1)
  - IEEE® 754 Compliant, Single-precision Floating-Point Unit (FPU)
  - DSP Instruction
  - Thumb-2 instruction set
  - Instruction and Data Cache Controller with 2 Kbytes of Cache Memory

- Segment LCD Controller
- AES and RSA Accelerators
- Integrity Check Monitor
  - Uses DMA to continuously compare checksums of memory regions



https://ww1.microchip.com/downloads/aemDocuments/documents/SE/ProductDocuments/DataSheets/SAM4C-Series-DS60001717.pdf

And the entire STM32 line

- Implementation License
-
- Power Efficiency and Cost optimized
- https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html