DHBW
Duale Hochschule
Baden-Württemberg
Heidenheim

Embedded-Entwicklung für RISC-Prozessoren am Beispiel ARM

ARM Business-Modell

- Die Firma ARM (brit.) stellt selbst keine Prozessoren/Controller her, sondern entwickelt nur sogenannte „IP-Cores“ , die von Herstellern wie Atmel, Infineon, ST, NXP, TI und vielen anderen lizenziert werden. Nvidia hat bis 2022 versucht, ARM (Eigentümer: Softbank, Japan) kaufen zu dürfen. Gescheitert...

- ARM ging im September 2023 public

- Die Chip-Hersteller ergänzen Speicher und Peripherie.

- Der Vorteil dieses Modells: sehr viele Prozessoren mit unterschiedlichster Ausstattung verfügbar; alle mit dem selben Befehlssatz (und damit dem selben Compiler) programmierbar.

ARM Business-Modell

- Allen ARM-Cores gemeinsam ist die **32/ 64 Bit RISC-Architektur**.

- Speicherplatz sparen: zusätzlich zum ARM-Befehlssatz (32bit-Instruktion) **Thumb**-Befehlssatz
  (bis auf einige Ausnahmen alle Befehle in 16bit codiert, langsamere Ausführungsgeschwindigkeit).

- Die **Cortex-M** Controller: ausschließlich Thumb-Instruktionen.
  Thumb-fähige Controller sind erkennbar am **T** in der Bezeichnung, z.B. ARM7**T**DMI.

- in 2021: 30 Mrd. Chips!

ARM-Prozessor-Profile

Client-Prozessor-Implementierungen:

- **Application Profile (‘A’):**
  High Performance, z.B. Mobile, Enterprise (betriebssystembasierte Anwendungen mit viel Leistung)

- **Real-Time Profile (‘R’):**
  Embedded Anwendungen, z.B. Automotive, Industriesteuerungen
  (sicheres Chip-Design für autonome Fahrzeuge und medizinische Geräte, Controller in Festplatten und Solid-State-Drives,
  in Fahrzeugen als Teil der Steuereinheit von Antiblockiersystemen oder in der Auslöseelektronik von Airbags)

- **Microcontroller Profile (‘M’):**
  Mikrocontroller für eine große Bandbreite an Anforderungen wie Kosten, Realtime, Performance,
  nicht zeitkritische steuer- und regeltechnischen Aufgaben (Geräte für‘s IOT, viele integrierte Funktionen).

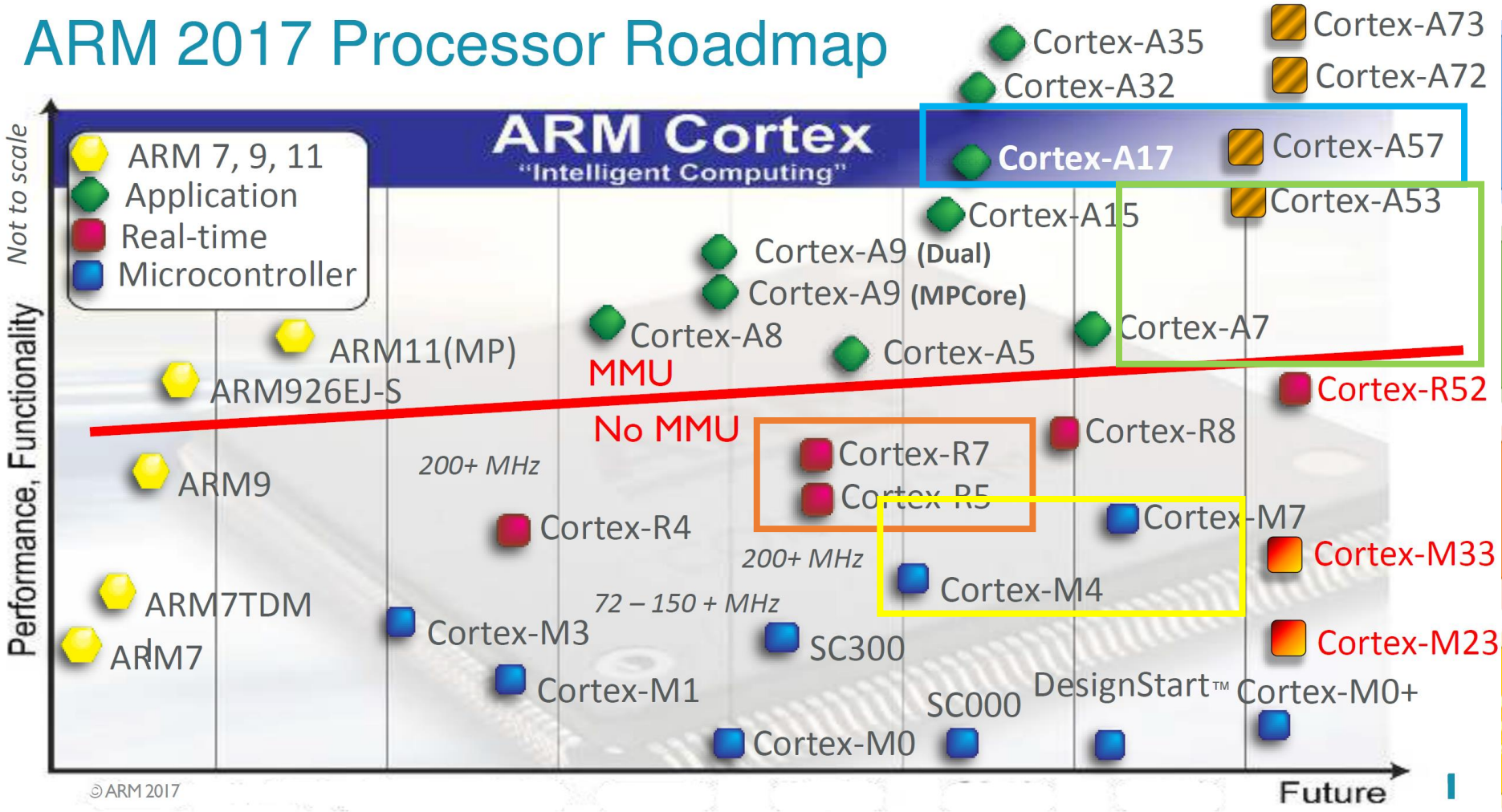
- **Custom Profile (‘X’):** Prozessorkerne nach Kundenwunsch

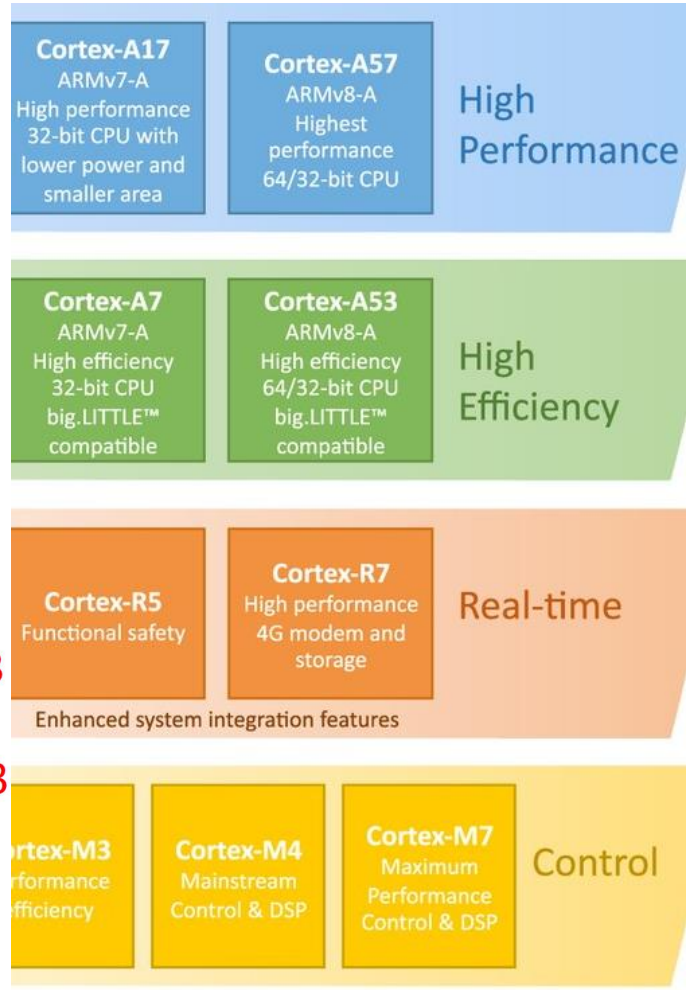
ARM-Prozessor-Profile

Server-Prozessor-Implementierungen:

- **Data efficiency (‚E'):** Befähigung der Infrastruktur, die Durchsatz-Bedürfnisse der nächsten Generation zu erfüllen, hoch skalierbarer Durchsatz für Datentransport vom Edge-Gerät zum Server
- **Performance/ Scalability/ 5G (‚N'):** Skalierbare Cloud to Edge-Infrastruktur, revolutionäre Rechenleistung
- **Machine Learning (‚V'):** „Graviton", Scalable vector extensions, höchste Performance
- **„SecurCore"-series:** Speziell designed für Hochleistungs-Smartcards und eingebettete Security-Applikationen, Cortex-M3 processor mit bewährten Sicherheitsfeatures

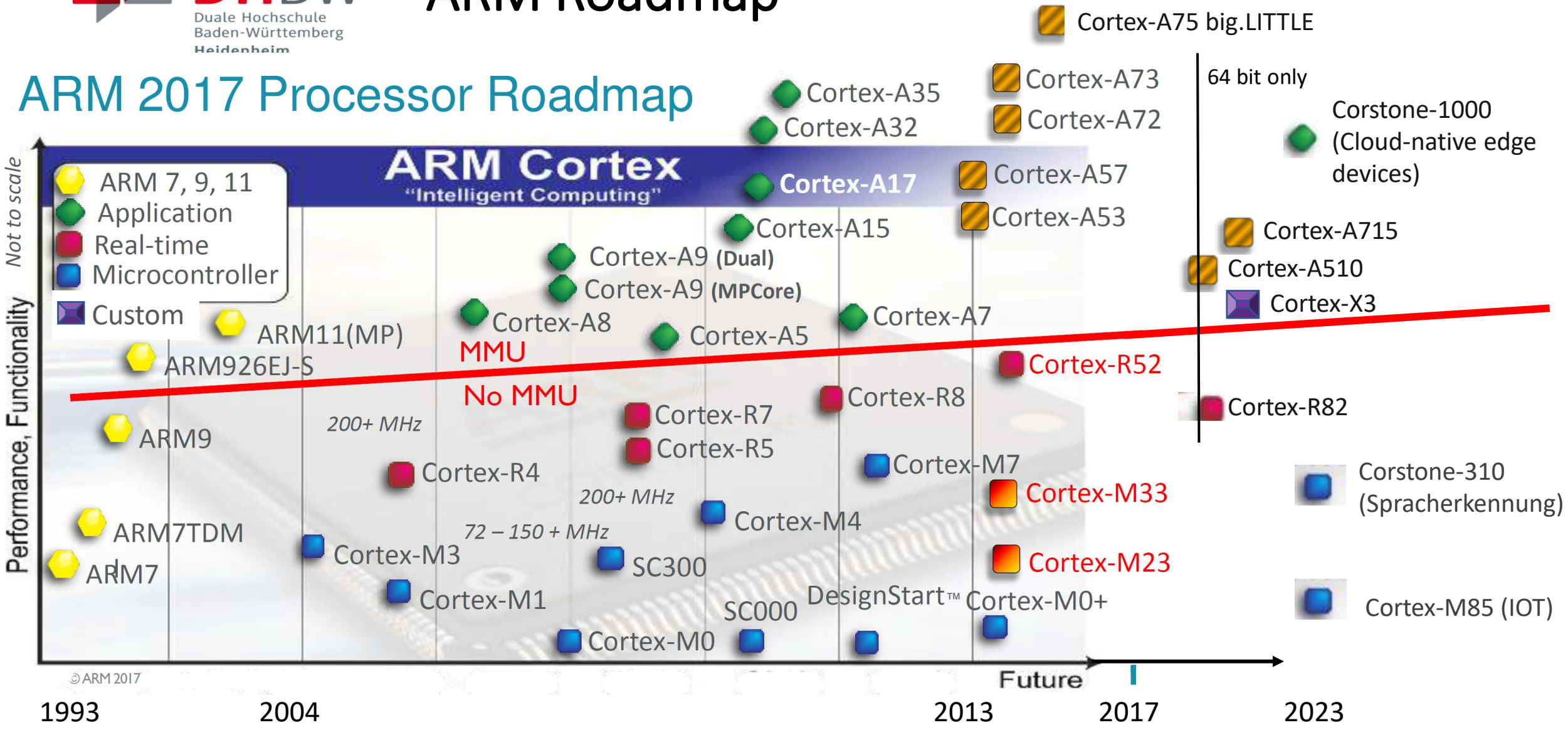
ARM Roadmap bis 2017





2017, R. Boys, Arm.com, https://www.google.de/search?hl=de&tbm=bks&q=arm+2017+prozessor+roadmap&spell=1&sa=X&ved=0ahUKEwiyyK3wtNPaAhVRC-wKHQvWAVwQBQgmKAA&biw=1280&bih=584&dpr=1.25

ARM Roadmap


ARM 2017 Processor Roadmap
ARM Cortex
"Intelligent Computing"
ARM 7, 9, 11
Application
Real-time
Microcontroller
Custom
Not to scale
Performance, Functionality
ARM7TDM
ARM7
ARM9
ARM926EJ-S
ARM11(MP)
Cortex-M3
Cortex-M1
Cortex-R4
200+ MHz
Cortex-A8
Cortex-A9 (MPCore)
Cortex-A9 (Dual)
Cortex-A35
Cortex-A32
Cortex-A17
Cortex-A15
Cortex-A5
MMU
No MMU
Cortex-R7
Cortex-R5
200+ MHz
72 – 150 + MHz
SC300
Cortex-M4
Cortex-M0
SC000
Cortex-A7
Cortex-R8
Cortex-M7
DesignStart™ Cortex-M0+
Cortex-A75 big.LITTLE
Cortex-A73
Cortex-A72
Cortex-A57
Cortex-A53
Cortex-R52
Cortex-M33
Cortex-M23
64 bit only
Corstone-1000
(Cloud-native edge devices)
Cortex-A715
Cortex-A510
Cortex-X3
Cortex-R82
Corstone-310
(Spracherkennung)
Cortex-M85 (IOT)
© ARM 2017
Future
1993
2004
2013
2017
2023

Aktuelle Roadmap: CPUs

Hauptaussage auf dem ARM DevSummit Okt. 2020:

- ab 2022 werden alle high-end (big) Arm CPU cores 64-bit-fähig sein (bisher unterstützen nur Cortex-A sowohl 32-bit and 64-bit)

- nicht nur die Adressierung von mehr Speicher, sondern die 64-bit-Befehle boosten die Performance um 15 to 30% verglichen mit den 32-bit-Befehlen).

ARM DevSummit 2022:

Session Track

→ IoT DevOps/Virtual Hardware

→ Cloud and 5G Infrastructure

→ Gaming

→ Windows on Arm ARM64EC (ABI, Microsoft)

→ Mobile Device Technology

→ Autonomous and Software-Defined Vehicles

ARM-Familien

Archi-tektur	ARM-Designs/ Familien	Release-jahr	(MHz)	Verwendung
ARMv1	ARM1	1985	4	BBC-Master
ARMv2	ARM2, ARM3	1986, 89	8...25	Acorn Archimedes
ARMv3	ARM6, ARM7	1991, 93	12...40	ARM Limited, z.B. Apple Newton
ARMv4	ARM7TDMI, ARM8, StrongARM, ARM9TDMI	1995 1997	16...75 203 180	...TDMI: SoCs für Mobiltelefone, Game Boy Advance, Nintendo DS (als Subprozessor) StrongARM: SoC mit DEC
ARMv5TE	ARM7EJ, ARM9E, ARM10E	1997	100... 1200	ARM9: getrennte Busse zu den Caches (Harvard) Intel-ARM9 (XScale) in PalmTungsten
ARMv6	ARM11(1176, 11MPCore, 1136, 1156) ARM Cortex-M (M0, M0+, M1)	2002	...1000 ...200	SIMD-Befehle, Multicore (Apple und Nokia Smartphones) Cortex: Mikrocontroller
ARMv7	ARM Cortex-A (A8, **A9**, A5, A15, A7, A12, A15) ARM Cortex-M (M3, M4, M7) ARM Cortex-R (R4, R5, R7, R8)	2004 2005 2011	bis 2500	SIMD-Einheit (NEON), primär für Multimedia, IPhone5s
ARMv8	ARM Cortex-A (A32, A53, A57, A72, A35, A73, A55, A75) ARM Cortex-M (M23, M33) ARM Cortex-R (R52)	2012 2016	1200-3000	64-Bit-Architektur
ARMv9	Arm Cortex-A (A510, A710, X2), Arm Neoverse (N2)	2021	3500	Kompatibilität zur 32-Bit-Architektur nur noch für A-Profil Server, 128 Kerne, 8xDDR5-Ctlr, Scalable Vector Extensions2 (SVE2), Windows-On-ARM (WOA) , Apple M1 (Mac)

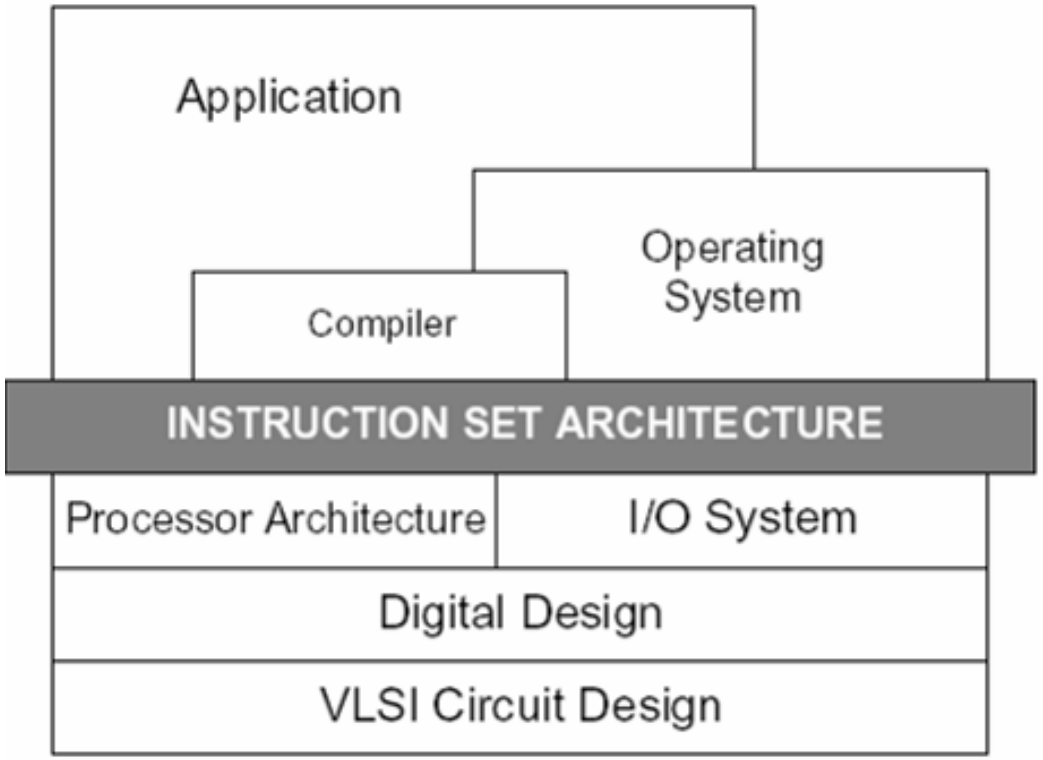
ARM-Prozessoren heute

Architektur-Versionen:

- Unterschiedliche Features
- z.B. unterschiedliche Multiplikationsbefehle, Adressierungsarten, etc.
- Jeweils gleiches Befehlssatzdesign für jede Version
- nach 2023 keine 32-Bit-Unterstützung mehr

Bedeutung des Befehlssatzes (ISA)

- Befehlssatz = Instruction Set Architektur (ISA)
- Der ISA Level ist das Interface zwischen der Software und der Hardware
- Der ISA Level definiert die Sprache, die sowohl von der Software als auch von der Hardware verstanden werden muss.



ARM Instruction Sets

Die ARM-Architektur unterstützt aktuell 3 Instruction Sets:

- A64 instruction set:
  eingeführt mit dem ARMv8-A, um die 64-bit Architektur zu unterstützen (Registerbreite, Adressen).
- A32 instruction set (auch „ARM“ genannt):
  32-bit Befehlssatz in pre-ARMv8-Architekturen
- T32 instruction set (auch „Thumb“ bzw. „Thumb2“ genannt):
  komprimierter 16/32-bit Befehlssatz in pre-ARMv8 Architekturen

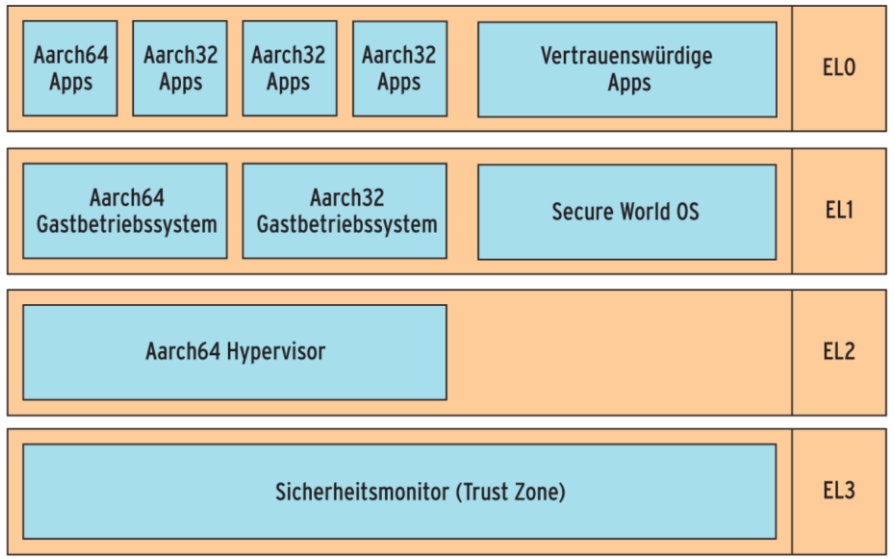
A64 und A32 haben eine feste Befehlslänge von 32 bit (A64: 32 Register, architektonische Änderungen).

T32 wurde als Ergänzung mit 16-bit Befehlen eingeführt, um eine bessere Komprimierung im User Code zu erreichen.

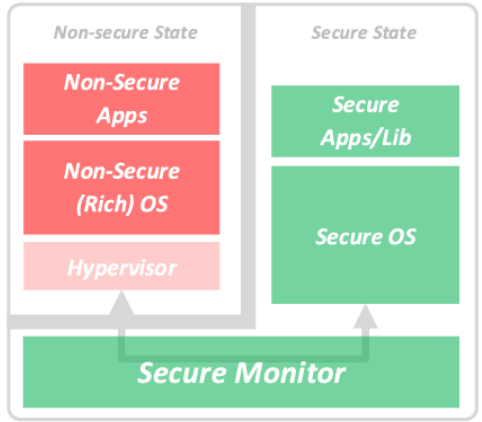
Über die Zeit hat sich T32 in ein gemischtes 16/32-bit instruction set entwickelt, das den Compilern ermöglicht, Performance und Codegröße auszubalancieren.

ARMv8, A64 Befehlssatz

- Prozessormodi (inkl. Ausnahmemodi) erneuert
- bedingte Ausführung der Befehle entfällt (für Out-of-order-Verarbeitung)
- Load-Store-Multiple nur noch mit 2 Regs
- 32-Bit-Modus arbeitet mit unterer Hälfte der Register (Bit 0...31),
  statt R0: W0 (64 bit) oder X0 (32 Bit)



Die Prozessormodi von ARMv8 und ihre Kompatibilität zu 32-Bit-Software. Die Ausnahme-Ebenen erinnern stark an die Ringe von x86 (Quelle: http://arm.com).



https://sweet.ua.pt/andre.zuquete/Aulas/AES/20-21/docs/Pinto19.pdf

Coding Standards UAL und APCS

Unified Assembler Language (UAL)

- Coding Standard, der von Compilern für ARM (32bit) und Thumb verstanden wird
- Macht Programme möglich, die mit verschiedenen Prozessoren kompatibel sind

ARM Program Calling Standard (AAPCS)

- Regelt den Aufruf von Unterprogrammen, Verwendung von Registern, etc.
- Ermöglicht Verwendung von Objektbibliothen, die mit anderen Compilern übersetzt wurden
- AAPCS64

Application Binary Interface (ABI)

- muss vom Executable erfüllt werden, um in einer bestimmten Umgebung lauffähig zu sein (z.B. Linux)
- relocatable Files müssen es erfüllen, um statisch linkbar und ausführbar zu sein

Architektonisches RISC-Erbe der ARM Prozessoren

Verwendete Merkmale:

- RISC
- Grundlegendes Design-Prinzip: Einfachheit
- Load/Store-Architektur (Register-Register), modif. Harvard-Architektur (L1-Cache ist Harvard)
- 32bit oder 64bit
- sowohl Little-Endian- als auch Big-Endian-kompatibel, kann also mit beiden Byte-Reihenfolgen umgehen (deutlicher Vorteil bei Einsatz als Standard-CPU in Kommunikationsgeräten)
- Der Standardmodus des ARM ist Little-Endian
- 3-Adressbefehle (32bit) oder gemischt (Thumb)

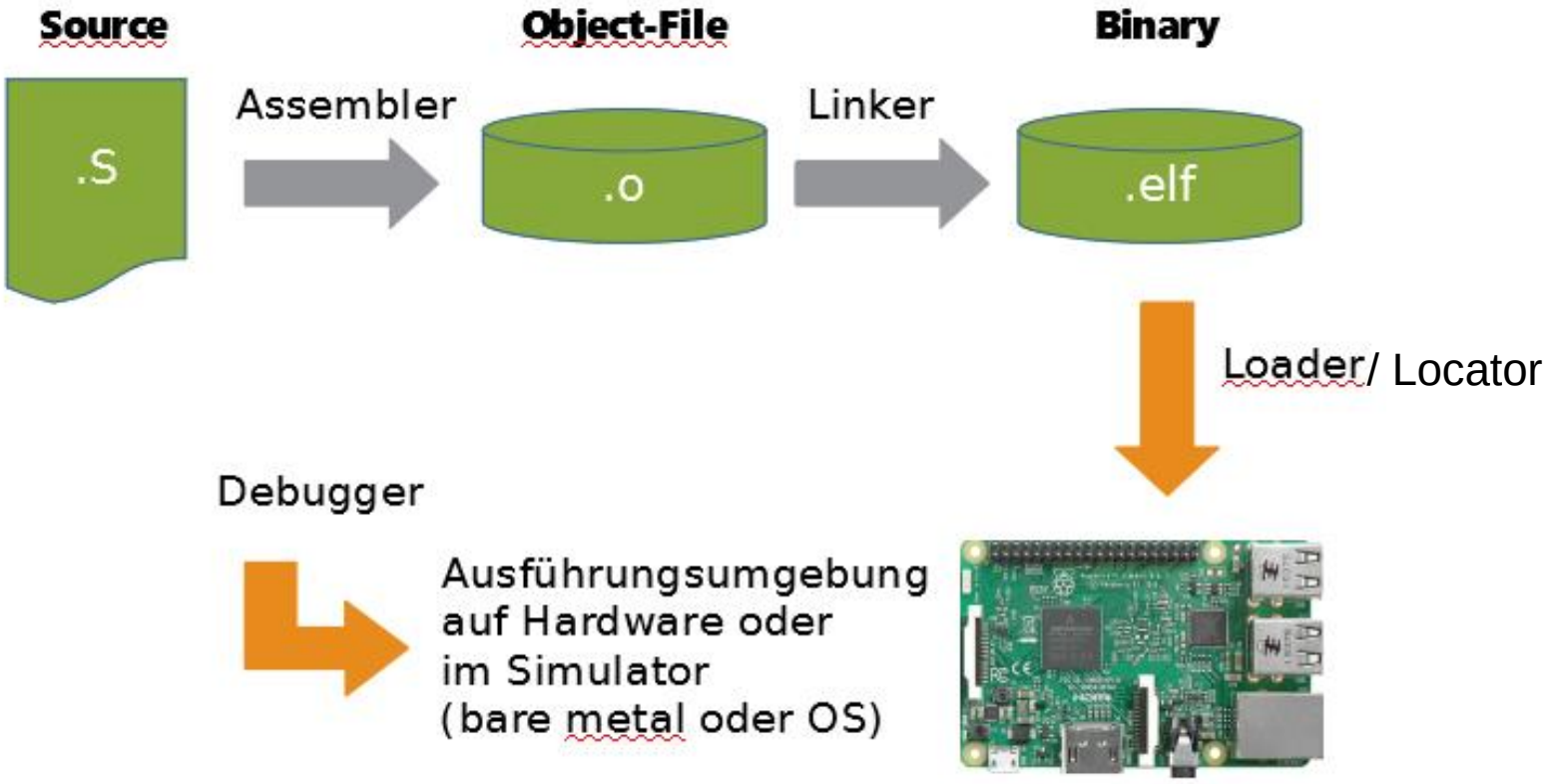
DHBW
Duale Hochschule
Baden-Württemberg
Heidenheim

Embedded Entwicklung für ARM

Ein erstes ARM-Programm (Gnu)

```
.file „add.S“                          @ filename
            .text                      @ code section
            .align   2                 @  lower 2 bits of each address are 0:
                                       @ 4 byte alignment (next instruction addr.: xxx00)
            .global  main              @ Label for debugger
            .type   main, function     @ for debugger
main:
            LDR       r0, number       @ r0=100
            ADD       r1, r0, #1       @  r1=101
            SUB       r2, r1, r0       @  r2=1
            MOV       pc, lr           @ return, besser: BX LR
number:  .word      100                @ constant, decimal
.lfe1:
            .size main,.lfe1-main      @ for linker, to calculate size of code
            .ident   „GCC: (GNU) 3.2.1“
   .end
```

Toolchain am Beispiel GNU ARM



Targets

Entwicklung auf dem Device

- Native Toolchain
- Benötigt ein OS

Cross-Entwicklung

- Compiler auf dem PC übersetzt für Hardware

Entwicklung im Simulator auf dem PC

- Compiler auf dem PC übersetzt für Hardware, ausgeführt im Simulator
- Wenn Hardware nicht verfügbar oder Entwicklung zu zeitaufwändig
- Insight, QEmu, Cpulator

Targets

Bare-metal

- Für Mikroprozessor ohne Betriebssystem

Betriebssysteme

- Linux
- Embedded Windows
- Realtime OS

Toolchain für das Praktikum (ARMv7)

IDE:	Cpulator (by Henry Wong)	
Assembler/ Linker/ Locater:	arm-elf-gcc	GNU ARM -> elf executable
Debugger/ Disassembler & UI:	Cpulator	
Debugger UI:	arm-elf-insight	GNU ARM https://www.mikrocontroller.net/articles/ARM-elf-GCC-Tutorial
Debugger:	arm-elf-gdb	GNU ARM

Bare Metal (ohne Betriebssystem), im Simulator interpretierend ausgeführt

GNU ARM Assembler

Format einer Befehlszeile:

[<label>:] [instruction or directive] @ comment

line comment

Instruktionen: Maschinenbefehle des Prozessors

Direktiven: Anweisungen an den Assembler (Direktiven), assemblerspezifisch (z.B. .word)!

GNU ARM Assembler

Adressierung, wobei r_n einen Registernamen angibt:

addr	Absolute Addressierung
r_n	Register-Adressierung
$[r_n]$	Indirekte oder indizierte Registeradressierung
$[r_n,\#n]$	Register-Adressierung mit Offset
#imm	Immediate Konstante dezimal (Bsp.: #16)
#&imm	Immediate Konstante hexadezimal (Bsp.: #&10, #0x10)
#0ximm	

ARM Pseudobefehle

- Steueranweisungen an den Compiler/ Assembler oder Linker
- Maschinensprache umfasst normalerweise nur eine kleine Anzahl an möglichen Instruktionen.
- Manche Operationen sind nicht direkt implementiert, sondern müssen durch andere realisiert werden.
- Einige Assembler unterstützen sogenannte Pseudobefehle. Dies sind Befehle, die es in der Maschinensprache nicht gibt, aber häufig gebraucht werden.
- Der Assembler übersetzt diese dann in eine geeignete Sequenz von „echten“ Assemblerbefehlen.
- Bsp: Laden eines Registers mit einer 32-Bit-Konstante

GNU ARM Assembler Direktiven: Sektionen

.text

legt einen Text-/ Code-Bereich an
inkl. Konstanten
(Flashbarer Speicherbereich)

.data

legt einen Datenbereich an (RAM) für initialisierte Daten (Initialisierungswerte und RAM)

.bss

zero-initialisierte Daten
(vom Dev uninitialisiert)

.global

nimmt ein Symbol in die globale Symboltabelle auf

.section

Fortsetzung einer anderen als der vorigen Section

niedere Adressen

höhere Adressen

GNU ARM Assembler Direktiven: weitere

.word Ausdruck

legt einen initialisierten Speicher an

.align #Bits

sorgt dafür, daß die nachfolgende Anweisung auf einer Speicherstelle steht, deren unterste # Bits 0 sind

.end

das Ende des Programms

.size nameFunction, .-nameFunction

berechnet die Größe einer Funktion, damit der Linker sie ausschließen kann, wenn nicht benutzt

.space #Bytes

reserviert 0-initialisierte Bytes in aktueller Sektion

.func name, [label]
.endfunc

wird vom Debugger genutzt

.type nameTable, object
.type nameFunction, function

Symboltyp für Assembler

.equ count, 5

Konstantendefinition als Makro

DHBW
Duale Hochschule
Baden-Württemberg
Heidenheim

ARM-Befehlssatz