

Echtzeitsysteme

Übungen zur Vorlesung

System-Software-Entwicklung

Simon Schuster Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Lehrstuhl für Informatik 4 (Verteilte Systeme und Betriebssysteme)
https://www4.cs.fau.de

Sommersemester 2022



1 Wiederholung: Stack-Aufbau

2 Standards in der Softwareentwicklung

3 Verwendung von Fließkommazahlen

4 Überblick: Toolchain

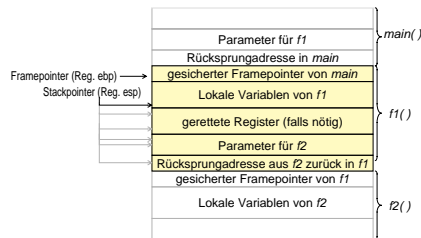
5 Hardware



Wiederholung: Stack-Aufbau

```

1 int main() {
2     int a, b, c;
3
4     a = 10;
5     b = 20;
6
7     f1(a, b);
8
9     return a;
10 }
    
```



Stack-Frame zur Verwaltung des Stacks

- Lokale Variablen
- Funktionsparameter
- Rücksprungadressen

Register zur Verwaltung des Stacks

- Stackpointer: Zeigt auf nächsten freien Speicherbereich.
- Framepointer: Zeigt auf Beginn des aktuellen Stackframes.



Ablauf Funktionsaufruf

```

int main(void) {
    int a, b, c;
    a = 10;
    b = 20;
    f1(a, b);
    return a;
}

int f1(int x, int y) {
    int i[3];
    int n;
    x++;
    n = f2(x);
    return n;
}

int f2(int z) {
    int m;
    m = 100;
    return z+1;
}

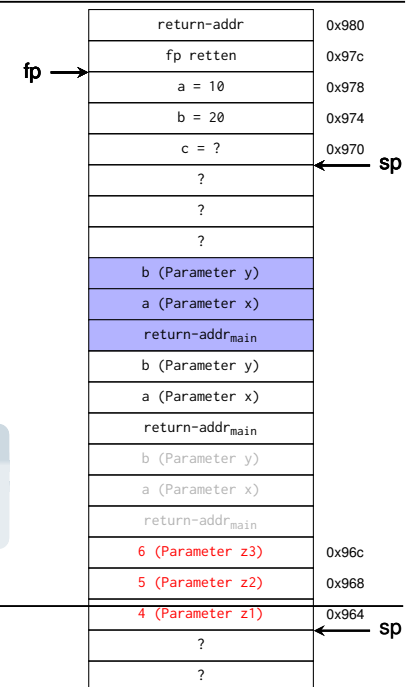
int main(void) {
    // ...
}

int f3(int z1, int z2, int z3) {
    int o;
    // ...
}
    
```

Stackframe abräumen

Was passiert bei weiterem Funktionsaufruf?

- fp = pop(sp)

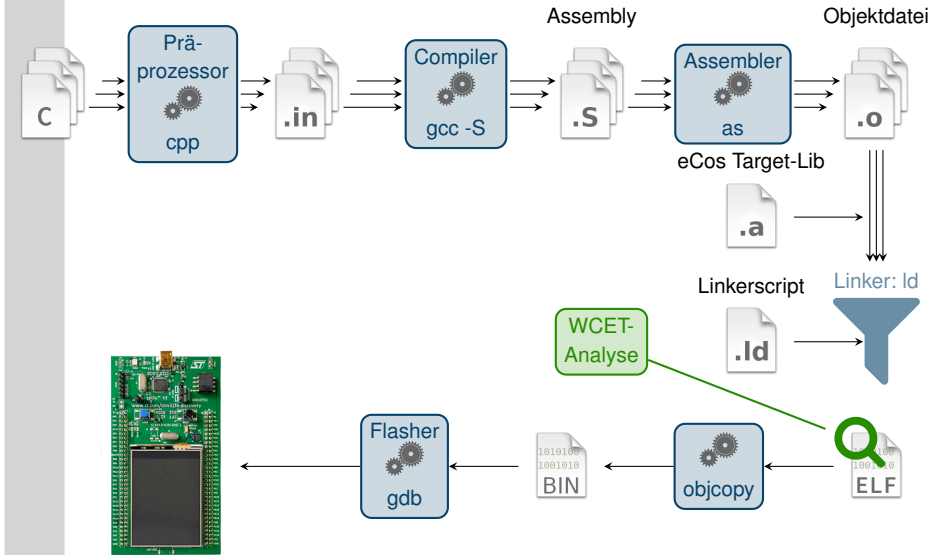


Übersicht

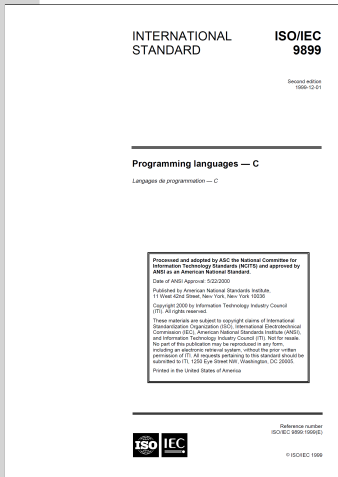
- 1 Wiederholung: Stack-Aufbau
- 2 Standards in der Softwareentwicklung
- 3 Verwendung von Fließkommazahlen
- 4 Überblick: Toolchain
- 5 Hardware



EZS-Toolchain



C Standard



- Mehrere Iterationen: C89, C99, C11, C18
- Früher ANSI, heute ISO/IEC Standards:
 - ANSI X3.159-1989
 - ISO/IEC 9899:1990
 - ...
- Unabhängiger Standard, von ISO entwickelt
- Beschreibt C Syntax & Semantik



C Standard II

6.5.5 Multiplicative operators

Syntax

multiplicative-expression:

cast-expression
multiplicative-expression * *cast-expression*
multiplicative-expression / *cast-expression*
multiplicative-expression % *cast-expression*

Constraints

Each of the operands shall have arithmetic type. The operands of the % operator shall have integer type.

3.4.3

undefined behavior

behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements

NOTE Possible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message), to terminating a translation or execution (with the issuance of a diagnostic message).

EXAMPLE An example of undefined behavior is the behavior on integer overflow.

Source: ISO/IEC 9899:TC3, S.4



Übersicht

- 1 Wiederholung: Stack-Aufbau
- 2 Standards in der Softwareentwicklung
- 3 Verwendung von Fließkommazahlen**
- 4 Überblick: Toolchain
- 5 Hardware



Frage #1

Zu was wird $7/2$ ausgewertet?

- 1 3.5
- 2 3**
- 3 nicht definiert in C

Erklärung

- Standard-Typ für Ganzzahlen ist `int`
- Rest verschwindet bei Ganzzahl-Division



Frage #2

Zu was wird $2/7$ ausgewertet?

- 1 1
- 2 0**
- 3 nicht definiert in C

Erklärung

- Standard-Typ für Ganzzahlen ist `int`
- Rest verschwindet bei Ganzzahl-Division



Frage #3

Zu was wird $7/2.$ ausgewertet?

- 1 immer noch 3
- 2 0
- 3 3.5**

Erklärung

- $2.0 == 2.$ \leadsto `double` auf der rechten Seite
- 7 wird in diesem Ausdruck als `double` behandelt, auch linke Seite
- Division zweier `double` Werte



Frage #5

Zu was wird $1/2 + 1/2$ ausgewertet?

- 1 nicht definiert
- 2 0
- 3 1 (dank Compileroptimierung)

Erklärung

- $int_1 / \langle \text{größerer } int_2 \rangle \rightsquigarrow 0 + 0 = 0$
- Compileroptimierung nicht C-Konform



Frage #6

Zu was wird $2 * M_PI$ ausgewertet?

- 1 6
- 2 ungefähr 6.28
- 3 6.283185307179586476925286766559005768394338798750...

Erklärung

- $M_PI \rightsquigarrow double$
- `double` Standard-Typ, außer zusätzliches Literal (3.14 f)
- Begrenzter Wertebereich:
6.283185307179586000000000000000



Frage #7

```

1 double a = 0.1;
2 double b = 0.2;
3
4 float aa = 0.1;
5 float bb = 0.2;
6
7 if (a+b == aa+bb){
8     ezs_printf("equal\n");
9 }else{
10    ezs_printf("unequal: %.30f != %.30f\n", (a+b), (aa+bb));
11 }

```

Was wird ausgegeben?

- 1 equal
- 2 unequal...



Fließkomma-Arithmetik

```

1 double a = 0.1;
2 double b = 0.2;
3
4 float aa = 0.1;
5 float bb = 0.2;
6
7 if (a+b == aa+bb){
8     ezs_printf("equal\n");
9 }else{
10    ezs_printf("unequal: %.30f != %.30f\n", (a+b), (aa+bb));
11 }

```

```

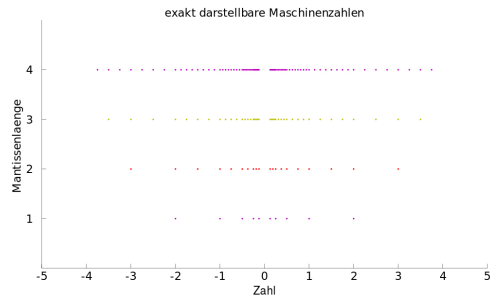
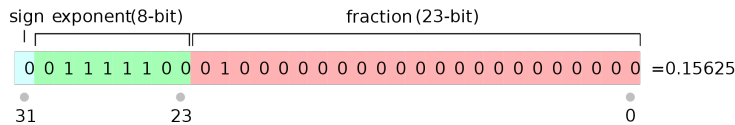
1 /* Ausgabe:
2 unequal:
3 0.300000000000000000000000000000 !=
4 0.300000011920929000000000000000
5 */

```

- Angenommen die Einheit ist Sekunden
 - 11,9 ns Fehler durch *einzelne Berechnung*
 - Kumulation der Rundungsfehler



Begrenzte Wertebereiche – IEEE 754



- IEEE 754
 - `sizeof(float) == 4`
 - `sizeof(double) == 8`



Probleme begrenzter Wertebereiche

- *What Every Computer Scientist Should Know About Floating-Point Arithmetic* [1]
- Rundungsfehler & Überläufe äußerst kritisch in *harten Echtzeitsystemen*
- Konvertierungen zwischen Größeneinheiten (`sec_to_nanosec: * 1e9`)
- Vermeidung des Wechsels von Größeneinheiten
- Verwendung von Festkomma-Arithmetik \rightsquigarrow VEZS
- Integer-Division ist *kein sicherer Ausweg*
- ➦ *Sorgfalt bei arithmetischen Operationen in begrenzten Wertebereichen*



Wahl des Datentyps bei Berechnung des Sinus-Wertes

- Harmonische Schwingung¹: $y(t) = y_0 \cdot \sin(\omega t + \varphi_0)$ und $\omega = 2\pi f$

```

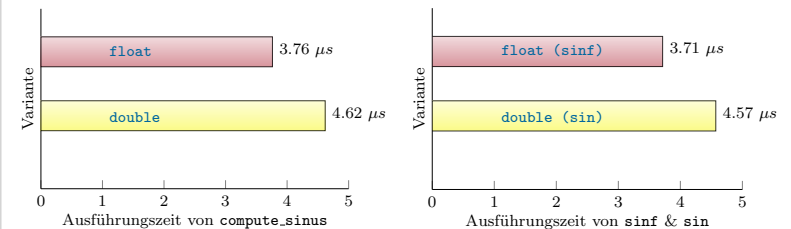
1 #define TYPE {int|double|float} ?
2 ...
3 TYPE compute_sinus(OTHER_TYPE real_time) {
4
5     TYPE f      = ...
6     TYPE omega = 2 * M_PI * f;
7     ...
8     ... sin(omega * real_time) // oder doch sinf(omega * real_time)?
9     ...
10 }
    
```

- *float oder double für Realzeit sinnvoll? Was ist OTHER_TYPE?*
- Konfiguration von `float` und `double` sinnvoll
- *Laufzeit von compute_sinus()?*



¹https://de.wikipedia.org/wiki/Schwingung#Harmonische_Schwingung

Vergleich der Laufzeiten



- Laufzeitzuwachs um 23% bei Wechsel `float` \rightarrow `double`
- Soft Float? Hard Float? hier: Soft Float
- Noch mehr Optimierungspotential? Wo wird die Laufzeit verbraucht?
 - 99% der Gesamtlaufzeit für `sinf` und `sin`
- Wahl des Datentyps in Abhängigkeit der Wortbreite (32-Bit Cortex-M4, 8-Bit AVR)
- Spezialbibliothek für Signalverarbeitung mit Integer-Arithmetik
- Spezielle Hardware-Einheiten zur Signalverarbeitung

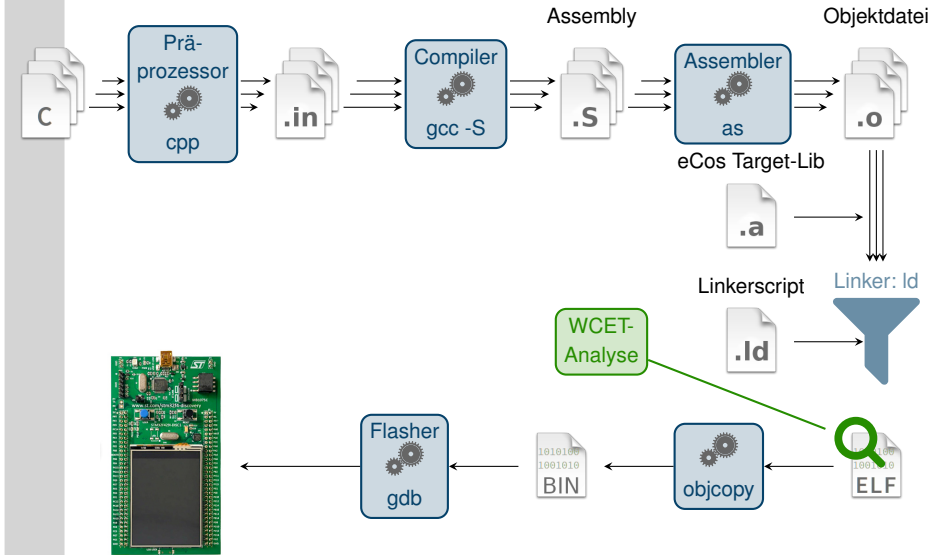


Übersicht

- 1 Wiederholung: Stack-Aufbau
- 2 Standards in der Softwareentwicklung
- 3 Verwendung von Fließkommazahlen
- 4 Überblick: Toolchain
- 5 Hardware



EZS-Toolchain



Präprozessor

Quellcode

```
1 #define F00 42
2
3 #include "example.h"
4
5 #if defined(F00)
6     int i = F00;
7 #else
8     int i = 0;
9 #endif
```

Expandiert

```
1 // Inhalt example.h
2 void example();
3
4 int i = 42;
```

Präprozessor

- Vorverarbeitungsschritt vor der Übersetzung
 - Konfigurationsabhängiger Code #if(def)
 - Definierbare Konstanten und Makros #define
 - Auflösen von #include-Direktiven
- Reine Zeichenersetzung/Textmanipulation



Übersetzer

Quellcode

```
1 volatile extern int i;
2 int j = 42;
3
4 int main(int argc, ...)
5 {
6     i = 0;
7     if(argc % 2) {
8         i = 1;
9     }
10    return i + j;
11 }
```

Assembly

```
...
ldr r3, [fp, #-8]
and r3, r3, #1
cmp r3, #0
beq .L2
ldr r3, .L4
mov r2, #1
str r2, [r3]
.L2:
...
```

Übersetzer

- Interpretation des Quelltexts gemäß Semantik laut Standard
- Umwandlung in Befehlssatz der Zielplattform
- Aufrufe gemäß Application Binary Interface (ABI)
- Optimierung des Kompilats



Beispiel: Schleifenaufrollen

Unoptimiert

```

1 for(i = 0; i < 40; i++) {
2     x++;
3 }
4 x++;
5 x++;
    
```

Größenoptimiert

```

1 for(i = 0; i < 42; i++) {
2     x++;
3 }
    
```

Laufzeitverhalten

- Optimierungen verändern Kontrollflussstrukturen
 - Schleifenaufrollen (siehe oben)
 - Schleifentauschen (loop interchange)
 - Schleifenneigen (loop skewing)
 - if-conversion
 - ...

~> invalidiert z.T. Annotationen und Annahmen über Laufzeitverhalten



Assembly

```

...
ldr r3, [fp, #-8]
and r3, r3, #1
cmp r3, #0
beq .L2
ldr r3, .L4
mov r2, #1
str r2, [r3]
.L2:
...
    
```

Objektdatei

```

...
e51b3008
e2033001
e3530000
0a000002
e59f3028
e3a02001
e5832000
...
    
```

Assembler

- Umwandlung der textuellen Repräsentation in Maschinencode (binär)
- 1:1 Übersetzung
- z.T. Macroassembler: Komplexbefehle zu Instruktionsfolge



Objektdatei

```

$ nm test.o
U i      # extern int i
00000000 D j
00000000 T main

$ nm i.o
00000004 C i      # Definition int i = 0
    
```

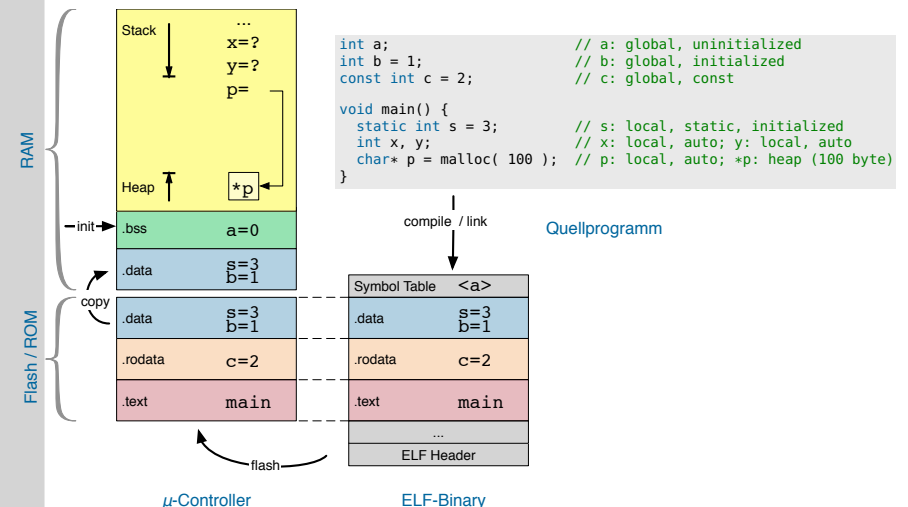
Binary

```

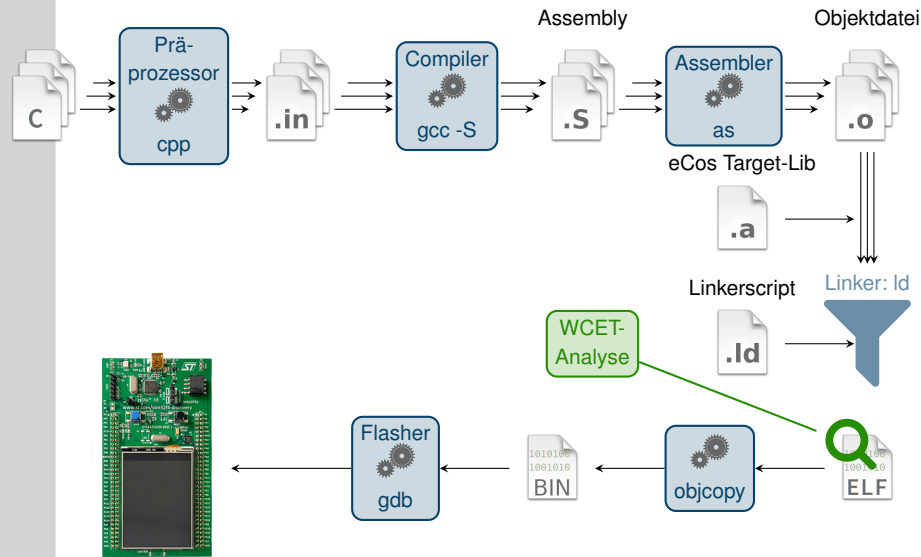
$ nm test.elf
00018a84 B i
00018634 D j
000081ec T main
...
    
```

Linker

- Variablen/Funktionen über Objektdateien verteilt
- ~> Zusammenführung der Funktionen und Variablen aus Objektdateien
- ~> Vergabe globaler Adressen gemäß Konfiguration
- ~> Auflösen der Adressen im Code



EZS-Toolchain



Instruktionssatz, Operationslaufzeiten

Logical	AND	AND Rd, Rn, <op2>	1
Divide	Signed	SDIV Rd, Rn, Rm	2 to 12 ^a
	Unsigned	UDIV Rd, Rn, Rm	2 to 12 ^a

3.3.1 Cortex-M4 instructions

The processor implements the ARMv7-M Thumb instruction set. Table 3-1 shows the Cortex-M4 instructions and their cycle counts. The cycle counts are based on a system with zero wait states.

Source: ARM, Cortex M4 Reference Manual r0p0, S.29

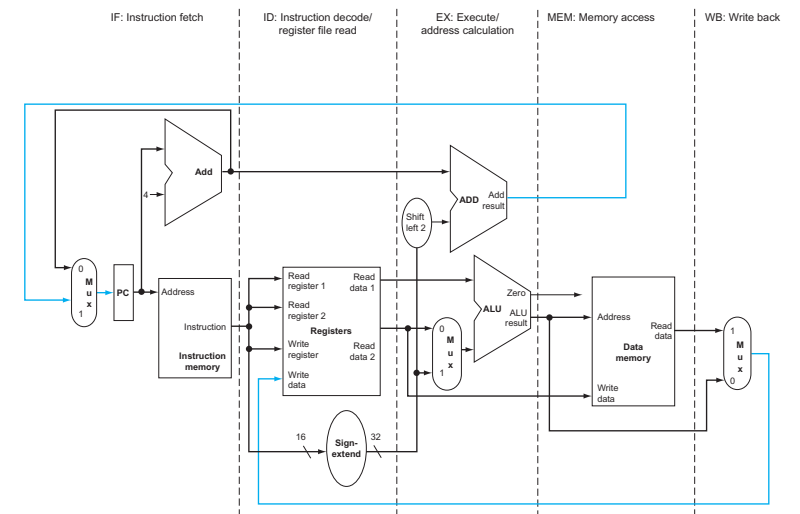
Instruktionslaufzeiten

- Zyklendauern aus Datenblättern
- Jedoch: Meist nicht vollständig
- Annahme hier: Zero-Wait-States ~ Kein Warten auf Speicher
- ~ Konkrete Hardwaremodellierung für jedes Board erforderlich

Übersicht

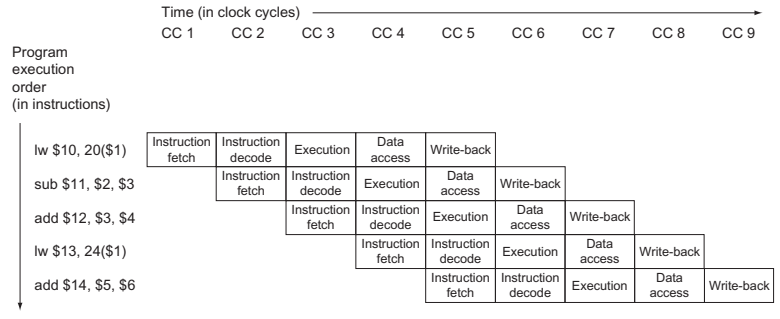
- 1 Wiederholung: Stack-Aufbau
- 2 Standards in der Softwareentwicklung
- 3 Verwendung von Fließkommazahlen
- 4 Überblick: Toolchain
- 5 Hardware

MIPS: Single-Cycle



Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012

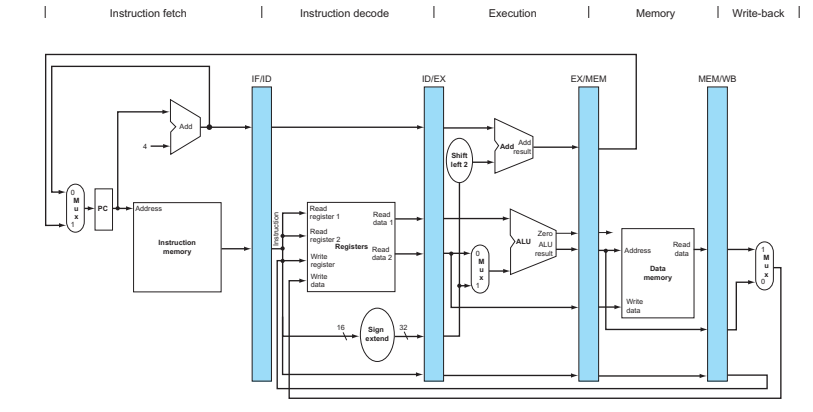
MIPS: Pipelining



Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012



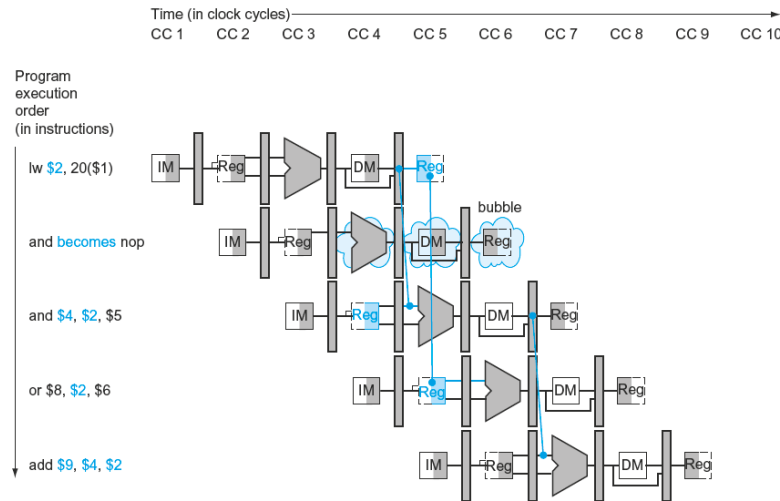
MIPS: Pipelining



Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012



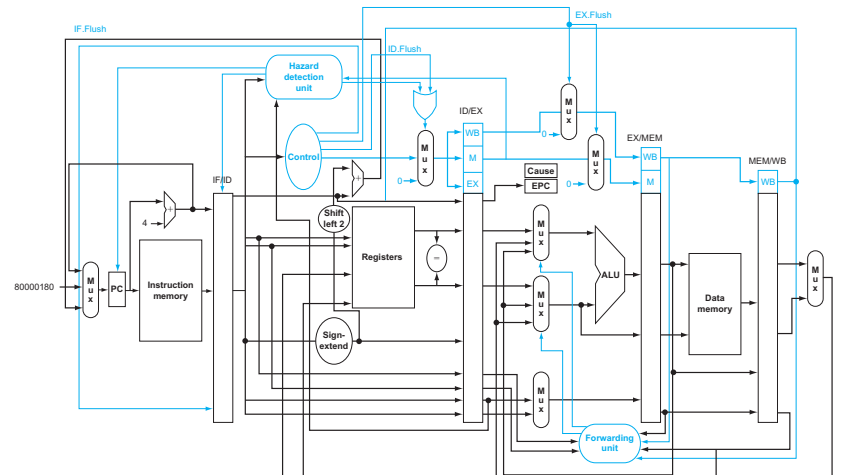
MIPS: Pipelining



Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012



MIPS: Pipelining



All dieses Wissen muss dem Analysetool bekannt sein

Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012

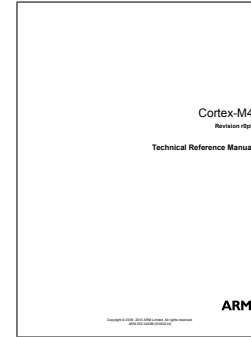


Eigenschaften von CPU-Architekturen

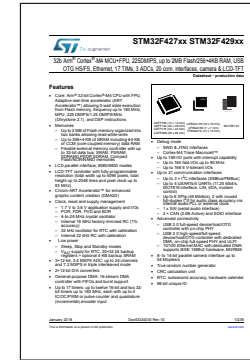
- Pipelining
- Caching
- Sprungvorhersage
- Mikroprogrammierbar vs. Fixed-Function
- Out-of-Order-Prozessoren
- Transaktionaler Speicher
- Superskalarität
- Mehrkernarchitekturen
- Hyperthreading
- ...
- ☞ All diese Funktionalitäten müssen dem Entwickler bekannt sein
- ☞ Berücksichtigung in der WCET-Analyse



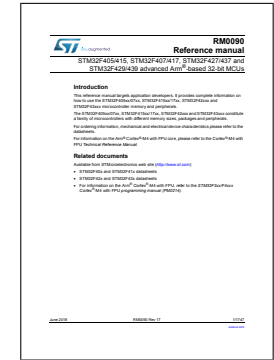
Referenzen



ARM: Cortex M4 –
Technical Reference
Manual
111 Seiten
Prozessorinterna



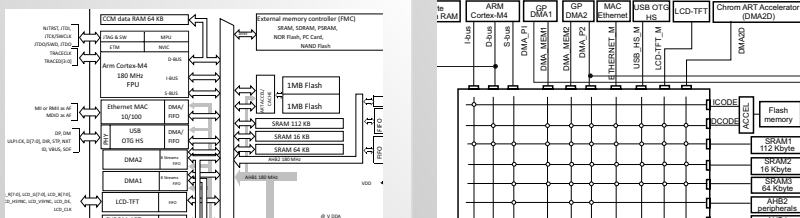
ST: STM32F427xx
STM32F429xx
Datasheet
240 Seiten
Boardspezifika



ST: RM0090
Reference manual
1747 Seiten
"Complete
Information on
STM32F4xxx"



Speichertopologie STM32F429i-DISC1



3.6 Embedded SRAM

- All devices embed:
- Up to 256Kbytes of system SRAM including 64 Kbytes of CCM (core coupled memory) data RAM
- DAM memory is accessed (read/write) at CPU clock speed with 0 wait states.

3.2 Adaptive real-time memory accelerator (ART Accelerator™)

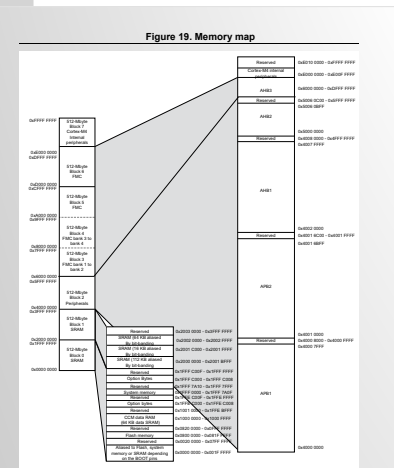
The ART Accelerator™ is a memory accelerator which is optimized for STM32 industry-standard Arm® Cortex®-M4 with FPU processors. It balances the inherent performance advantage of the Arm® Cortex®-M4 with FPU over Flash memory technologies, which normally requires the processor to wait for the Flash memory at higher frequencies.

To release the processor full 225 DMIPS performance at this frequency, the accelerator implements an instruction prefetch queue and branch cache, which increases program execution speed from the 128-bit Flash memory. Based on CoreMark benchmark, the

Source: ST: STM32F427xx STM32F429xx Datasheet, S.21



Speicherlayout STM32F429i-DISC1



Source: ST: STM32F427xx STM32F429xx Datasheet, S.53

APB2	
0x4001 3800 - 0x4001 38FF	SYSCFG
0x4001 3400 - 0x4001 37FF	SPi4
0x4001 3000 - 0x4001 33FF	SPi1
0x4001 2C00 - 0x4001 2FFF	SDIO
0x4001 2400 - 0x4001 2BFF	Reserved
0x4001 2000 - 0x4001 23FF	ADC1 - ADC2 - ADC3
0x4001 1800 - 0x4001 1FFF	Reserved
0x4001 1400 - 0x4001 17FF	USART6
0x4001 1000 - 0x4001 13FF	USART1

Source: ST: STM32F427xx STM32F429xx Datasheet, S.55

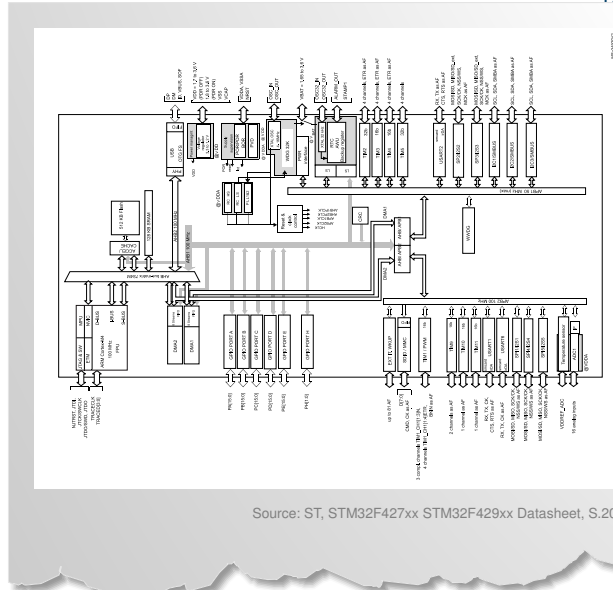
Peripherie

- Im Adressraum eingeblenet
 - Am Peripheriebus (APBx)
- ↪ Anderes Zugriffsverhalten als Speicher



Beispiel: USART

Blockdiagramm

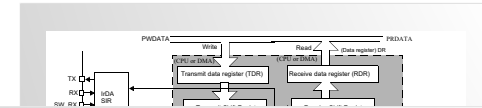


Source: ST, STM32F427xx STM32F429xx Datasheet, S.20



Beispiel: USART

Innerer Aufbau



30.6.1 Status register (USART_SR)

Address offset: 0x00

Reset value: 0x0000 00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
Reserved						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

Bit 7 TXE: Transmit data register empty

This bit is set by hardware when the content of the TDR register has been transferred into the shift register. An interrupt is generated if the TXEIE bit = 1 in the USART_CR1 register. It is cleared by a write to the USART_DR register.

0: Data is not transferred to the shift register

1: Data is transferred to the shift register

Note: This bit is used during single buffer transmission.

Source: ST; RM0090 Reference manual, S.1007 & 1008



Board Support Package

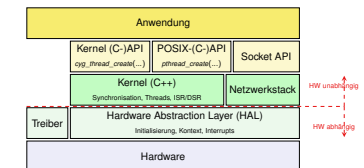
```
stm32f411e-discovery
|-- Release_Notes.html
|-- stm32f411e_discovery_accelerometer.c
|-- stm32f411e_discovery_accelerometer.h
|-- stm32f411e_discovery_audio.c
|-- stm32f411e_discovery_audio.h
|-- STM32F411E-Discovery_BSP_User_Manual.chm
|-- stm32f411e_discovery.c
|-- stm32f411e_discovery_gyroscope.c
|-- stm32f411e_discovery_gyroscope.h
'-- stm32f411e_discovery.h
```

Board Support Package

- Vom Hersteller vorgegeben
- Ansteuerung für Boardperipherie
- Meist permissive Lizenzen



Betriebssystem

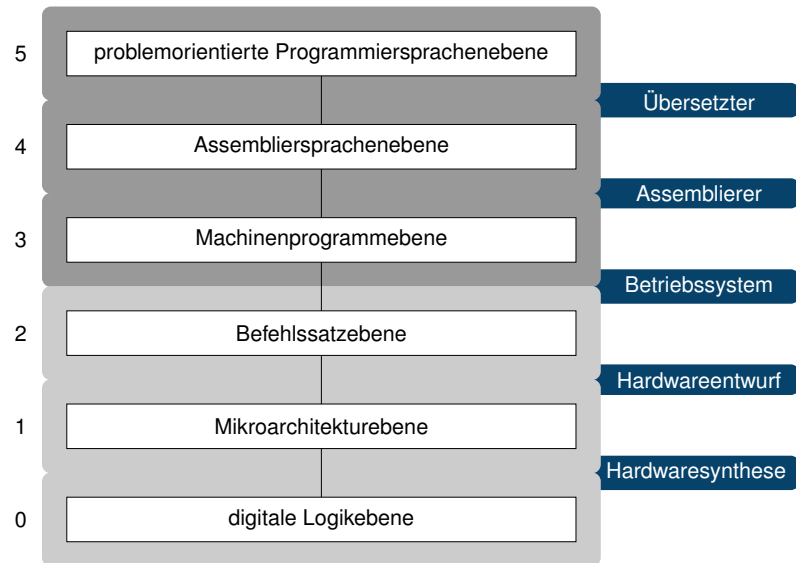


Betriebssystem

- in jedem Fall Ablaufplaner
 - oft Treiber/BSP mitgeliefert
 - ggf. interne Kontrollflüsse/Fäden/Unterbrechungen
 - meist konfigurierbar
- ~> Großer Einfluss auf Zeitverhalten des Gesamtsystems



Ebenen



Fazit

- Systemsoftwareentwicklung benötigt holistisches Wissen über
 - Werkzeugkette
 - Betriebssystem
 - Zielarchitektur
 - Echtzeittheorie
- ↪ Umfasst Interna, nicht immer verfügbar
- Entwickler muss all diese Einflussfaktoren kennen:
 - Zur Entwicklung
 - Zur Analyse
- ↪ Annahmen durch statische Analyse kontinuierlich verifizieren
- ↪ Nur so erhalten wir ein **sicheres** Echtzeitsystem



Literatur

- [1] David Goldberg.
What every computer scientist should know about floating-point arithmetic.
ACM Computing Surveys (CSUR), 23(1):5–48, 1991.

