# Betriebssysteme
## Übungen

Barry Linnert

**Wintersemester 2017/18**

# Was ist ARM?



- Eine Firma
  - früher: Advanced RISC Machine Ltd.
  - jetzt: ARM Ltd.

- Ein Prozessordesign
- Eine Bezeichnung für Prozessorkerne wie bspw. ARM920T

- Ein Geschäftsprinzip
  - keine eigene Herstellung der Hardware
  - verkauft werden nur Designs von Prozessoren in Form von Hardwarebeschreibungen
  - Lizenznehmer wie Atmel integrieren Prozessorkerne in eigene Hardware wie bspw. AT91RM9200

# Einordnung

- Zielsysteme sind eingebettete Systeme
  - geringer Energieverbrauch
  - hohe Integrationsdichte (System-on-Chip)
  - hohe Performanz (im Vergleich zu anderen Mikrocontrollern)
  - Fähigkeit »übliche« Betriebssysteme auszuführen
- Performanz
  - nicht direkt mit PC-Technik vergleichbar
  - bei Integerverarbeitung entspricht Atmel AT91RM9200 ungefähr einem Pentium 1 bei gleicher Taktfrequenz
- Energieverbrauch
  - AT91RM9200 benötigt unter Vollast 25mA bei 1,8 V: 45mW
  - AT91RM9200 benötigt in Standby 0,5mA bei 1,8 V: 0,9mW
  - üblicherweise sind keine besonderen Kühlmaßnahmen nötig

# Befehlssätze



- Befehlssätze beginnen mit ARMv<version> gefolgt von der Aufzählung der Varianten
- Aber: alle Modelle ab v4 unterstützen in der Regel M, so dass dies nicht mehr aufgeführt wird,
- stattdessen werden Modelle ohne M als xM gekennzeichnet (z.B. ARMv5TxM)

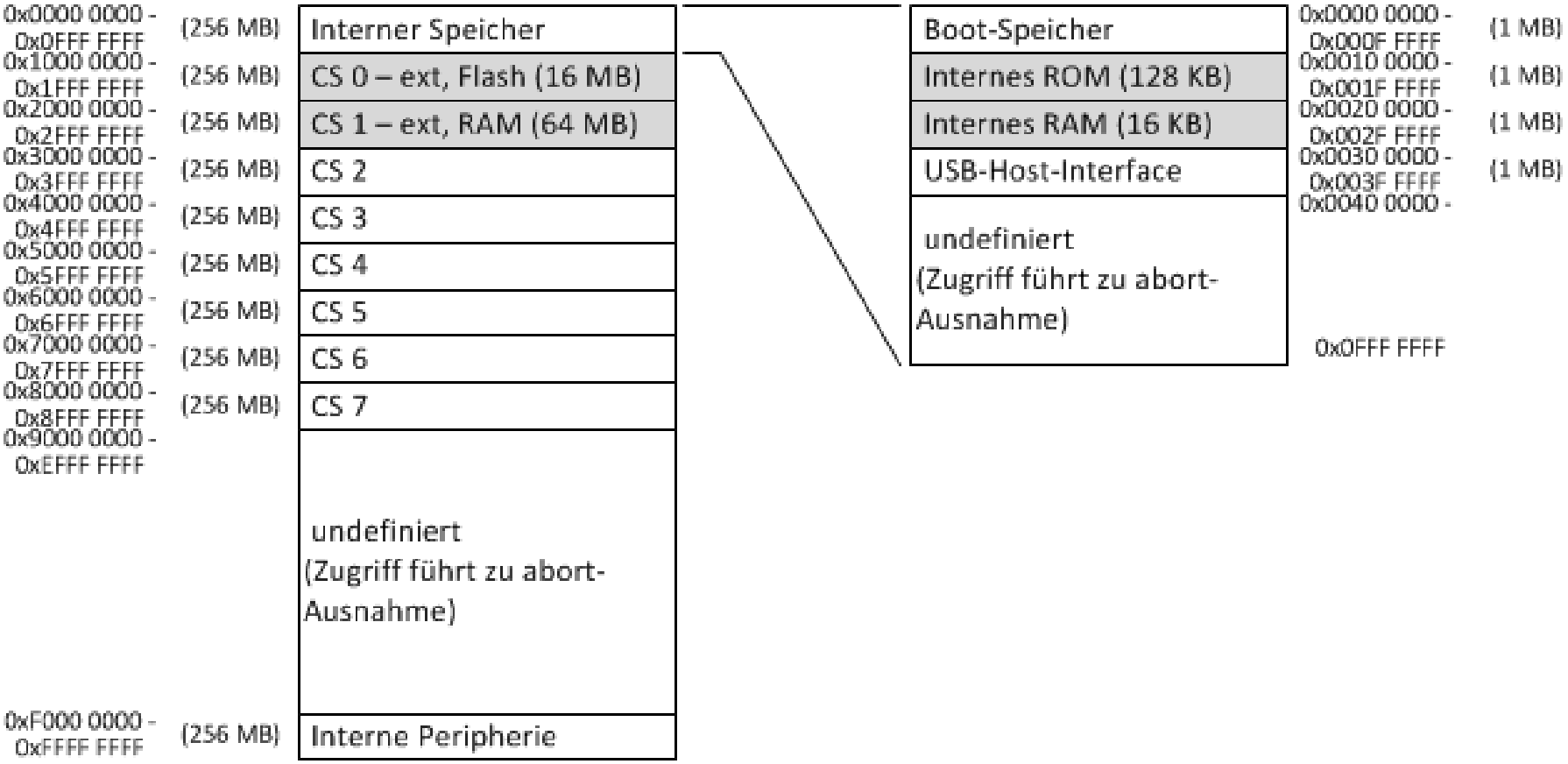
- **ARMv1** nie in kommerziellem Produkt verwendet, 26-Bit Adressraum, relativ schmaler Befehlssatz
- **ARMv2** erweitert ARMv1, 26-Bit Adressraum, mehr Register, unterstützt Koprozessoren und Multiplikationsinstruktionen
- **ARMv3** 32-Bit Adressraum, führt zwei Statusregister, zwei neue Modi und lange Multiplikation (M-Variante) ein
- **ARMv4** neuer Prozessormodus (System), unterstützt Thumb-Betriebsart (T-Variante), keine 26-bit Adressierung mehr
- Der Befehlssatz der Zielplattform ist ARMv4T, womit alle in der Vorlesung vorgestellten Details ab dieser Version gültig sind.

# Parameter der Architektur



- RISC-Design, Load-/Store-Architektur
  - 32-Bit Adress- und Datenbus (max. 4 GiB adressierbar)
- 3-Operanden-Instruktionen mit fester Länge
  - 49 ARM-Instruktionen, je 32 Bit breit
  - 35 Thumb-Instruktionen, je 16 Bit breit
- 7 Prozessormodi, insgesamt 37 Register
- Datentypen
  - Wort (32 Bit), an 4-Byte-Grenzen ausgerichtet
  - Halbwort (16 Bit), an 2-Byte-Grenzen ausgerichtet
  - Byte (8 Bit)
- Memory Management Unit (MMU) optional
  - ARM920T hat eine MMU

# Speicherorganisation AT91RM9200



# Speicherorganisation AT91RM9200

- 32 Adressleitungen gliedern sich in
  - 4 Bit (31..28) für Auswahl einer von 16 Regionen
  - 28 Bit (27..0) für die Adressierung innerhalb der 256 MiB Region
  - (ermöglicht einfache Integration unterschiedlicher Speicher und Peripherie)
- Regionen
  - Region 0: interne Speicher
  - Region 1–8: nach außen geführte Chip Select Bereiche CS0–CS7
  - Region 9–14: nicht definiert (Zugriff führt zu Abort-Ausnahme)
  - Region 15: Peripherie
- Zugriff auf Chip Select Region generiert »Chip Select«-Signal, auf das genau eine Komponente reagiert (z. B. RAM, ROM, Display)
- Hat eine Komponente weniger Speicher als adressierbar ist, werden i. d. R. die oberen Adressleitungen ignoriert →

  Wrap-Around

# Speicherorganisation AT91RM9200

- Interne Speicher (Region 0) nochmals untergliedert (Bit 27..20)
  - interne Region 0: Boot-Speicher
  - interne Region 1: 128 KiB internes ROM
  - interne Region 2: 16 KiB internes RAM
  - interne Region 3: USB-Interface
- Adressen im Boot-Speicher werden ausgeführt/angesprungen bei Reset und anderen Ausnahmen
- Boot-Speicher ist über Memory-Controller und externe Leitung BMS konfigurierbar
  - ohne Remap: externe Leitung »Boot Mode Select« (BMS) bestimmt, welcher nicht-flüchtige Speicher über Boot-Speicher zugreifbar ist (zum Booten; bei uns: BMS = 0)
    - BMS = 0 ! CS0 (externes Flash)
    - BMS = 1 ! internes ROM
  - mit Remap: internes RAM ist über Boot-Speicher zugreifbar (für Ausnahmebehandlung)

# Prozessormodi

Freie Universität Berlin

| Modus | Kurz-form | Modus-bits | Beschreibung |
|---|---|---|---|
| User | usr | 10000 | regulärer unprivilegierter Modus |
| FIQ | fiq | 10001 | Behandlung von *Fast Interrupt Requests* |
| IRQ | irq | 10010 | Behandlung von normalen *Interrupt Requests* |
| Supervisor | svc | 10011 | Behandlung von Systemrufen/Systeminitialisierung |
| Abort | abt | 10111 | Behandlung von Speicherzugriffsfehlern |
| Undefined | und | 11011 | Behandlung von unbekannten Instruktionen |
| System | sys | 11111 | regulärer privilegierter Modus |

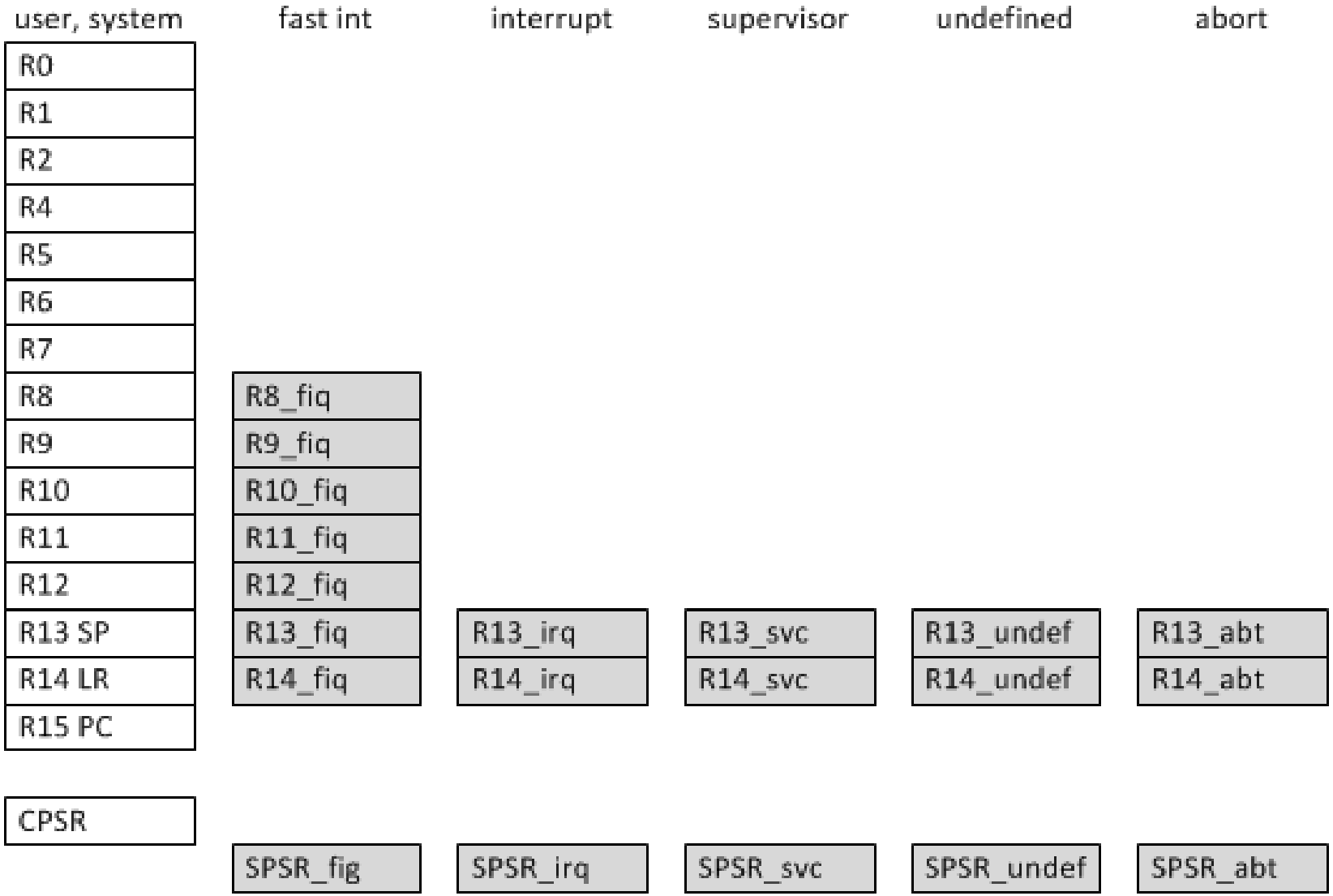
- User-Modus ist einziger nicht privilegierter Modus
- FIQ, IRQ, Supervisor, Abort und Undefined sind Ausnahmemodi
- Prozessor startet im Supervisor-Modus
- privilegierte Modi können in beliebige andere Modi wechseln; aus dem User-Modus kommt man nur durch Ausnahmen heraus

# Register

- Für Nutzer sind immer R0–R15 und CPSR sichtbar
  - Register R0–R12 sind beliebig nutzbar
  - R13: SP – Stack Pointer (meistens)
  - R14: LR – Link Register (Rücksprungadresse bei Ausnahmen und Funktionsaufrufen)
  - R15: PC – Program Counter
  - CPSR – Current Program Status Register
  - SPSR – Saved Program Status Register

- Ausnahmemodi überlagern einen Teil der Register mit eigenen Registern
  - erzeugt modusabhängige Sicht auf Register
  - vereinfacht den Moduswechsel: weniger Register zu sichern
  - muss bei Betriebssystementwicklung beachtet werden

# Modusspezifische Registersätze



# Modusspezifische Registersätze

Freie Universität Berlin

- Zusätzliche Register in FIQ erlauben sehr schnelle Behandlung von (Fast) Interrupts
- Jeder Ausnahmemodus hat eigene Register für Stack, LR und SPSR
  - eigenes LR/SPRS notwendig aufgrund potentiell geschachtelter Ausnahmen
  - getrennte Stacks erlauben besser vorhersagbares Verhalten
  - gemeinsame Stacks möglich, aber mit höherem Programmier- und Laufzeitaufwand verbunden
- Position der Stacks kann beliebig verändert werden
  - Alignment und Wachstumsrichtung beachten!
  - Nur ändern, wenn Stack ungenutzt!
  - Es kann immer nur der Stack des aktuellen Modus geändert werden!

# Prozessor-Statusregister


- Bit 31..28 sind Bedingungsbits
  - negative, zero, carry, overflow
  - werden für bedingte Sprünge und bedingte Ausführung genutzt
- Bit 7..0 sind Kontrollbits (PSR_c)
  - I, F maskieren Interrupt- und Fast-Interrupt-Signale
  - T zeigt Thumb-Modus an
  - $M_4$..$M_0$ kodieren den Prozessormodus
  - (Kontrollbits können in privilegiertem Modus gesetzt werden; extra Befehle für T-Bit)

# Ausnahmen

- Es gibt 7 Ausnahmen, jede Ausnahme veranlasst den Prozessor:
  - die aktuelle Ausführung zu unterbrechen,
  - in einen der 5 Ausnahmemodi zu wechseln,
  - und die Ausführung anderswo fortzusetzen.

- Prozessoraktionen bei Auftreten einer Ausnahme

```
R14_<exception_mode> = return link
SPSR_<exception_mode> = CPSR

CPSR[4:0] = exception mode number
CPSR[5] = 0        /* execute in ARM state */
if <exception_mode> = RESET or FIQ
CPSR[6] = 1        /* disable FIQ */
CSPR[7] = 1        /* disable IRQ */

PC = exception vector address
```

# Ausnahmen

Freie Universität Berlin

| Ausnahme | Modus | PC |
|---|---|---|
| Reset | svc | 0x0000 0000 |
| Undefined Instruction | und | 0x0000 0004 |
| Software Interrupt (SWI) | svc | 0x0000 0008 |
| Prefetch Abort | abt | 0x0000 000C |
| Data Abort | abt | 0x0000 0010 |
| IRQ | irq | 0x0000 0018 |
| FIQ | fiq | 0x0000 001C |

# Ausnahmen

- Behandlung von Ausnahmen
  - ggf. Register sichern
  - Grund für Ausnahme feststellen (wenn mehr als einer in Frage kommt)
  - Grund entsprechend behandeln, ggf. Rücksprungadresse modifizieren
  - ggf. Register wieder herstellen
  - Fortsetzung der eigentlichen Ausführung
- Rücksprung aus Ausnahmebehandlung
  - Modus zurücksetzen (SPSR → CPSR)
  - Zurückspringen (LR → PC)
- Die Reset-Ausnahme stellt hierbei einen Spezialfall dar, da sie genutzt wird, um ein eingebettetes System zu initialisieren → Keine Rückkehr!

# Ausnahmen

- Bei ARM ist die Interrupt Vektor Tabelle (IVT) hart verdrahtet: 0x0000 0000 – 0x0000 001C
  - Umstellung auf *high exception vectors* möglich:
    0xFFFF 0000 – 0xFFFF 001C
- Problem: Bei allen Ausnahmen (außer FIQ) steht für den Handler nur ein einziges Wort (32 Bit) zur Verfügung → Platz für genau einen (Sprung-)Befehl

- Übliche Lösungen
  - Sprung relativ zu PC (beschränkte Sprungweite)
  - Absoluter Sprung (beschränktes Alignment der Zieladresse)
  - Absoluter Sprung mit Zieladresse hinterlegt relativ zu PC (benötigt Extra-Wort in der Nähe)
  - sofortiger Rücksprung

# Ausnahmen

- Bei gleichzeitigem Auftreten mehrerer Ausnahmen entscheidet Priorität über Auslösereihenfolge:
  1. Reset
  2. Data abort
  3. FIQ
  4. IRQ
  5. Prefetch Abort
  6. Undefined Instruction/Software Interrupt

     (eine Instruktion ist entweder das eine oder das andere)

- Ausnahmen können durch Ausnahmen unterbrochen werden
  - IRQ und FIQ können gesperrt werden (und werden z. T. automatisch gesperrt)
  - Achtung: Behandlungsroutinen müssen entsprechend programmiert sein

# Assembler

- Wir entwickeln ein Betriebssystem! Komplett ohne Assembler funktioniert es nicht!
- Grundsätze
  - C für alles, was in C machbar ist (ohne Hacks)
  - Assembler für maschinennahe Operationen
  - Herausforderung besteht in der richtigen Aufteilung
  - Fokus auf Verständnis, nicht auf Optimierung
- Pro Assembler
  - Volle Kontrolle über Resultat
  - Alle unterstützten Instruktionen stehen zur Verfügung
- Contra Assembler
  - Spaghetticode, i. d. R. schwerer lesbar
  - keine Typprüfung
  - Plattform-spezifisch
  - keine Optimierung durch den Compiler

# Procedure Call Standard for the ARM Architecture (AAPCS)

Freie Universität Berlin

- Mischung von C und Assembler erfordert die Einhaltung von Konventionen
  - GCC hält sich an AAPCS
- Procedure Call Standard for the ARM Architecture (vereinfacht)
  - Parameter 1 bis 4 sind in R0 bis R3, weitere Parameter auf dem Stack
  - Stack ist *full-descending*: SP/R13 zeigt auf jüngstes Element und wächst Richtung Null
  - SP/R13 ist an 8 Byte Grenze ausgerichtet
  - LR/R14 beinhaltet die Rücksprungadresse
  - R4–R11 und SP/R13 müssen bei Rückkehr gleiche Werte enthalten wie vorher → R0–R3, R12, LR/R14 und Bedingungsflags können beliebig verändert werden
  - Rückgabewert wird in R0 transportiert
- (Für weitere Details, etwa 64-Bit-Argumente, siehe Standard.)

# Nutzung von Assembler aus C heraus



- Assembler-Seite
  - Label als global deklarieren
  - an AAPCS halten
- C-Seite
  - Funktionsprototypen für Assembler-Funktionen
  - Assembler-Daten als extern deklarieren
  - Achtung: Es findet keinerlei Prüfung statt, ob die Deklarationen korrekt sind!

bar.S

```
.section .text
.global add
add:
        add r0, r0, r1
        mov pc, lr

.section .data
.global message
message:
        .asciz "Hallo Welt\r\n“
```

bar.h

```
#ifndef _BAR_H_
#define _BAR_H_

int add(int a, int b);
extern char message[];

#endif /* _BAR_H_ */
```

# Nutzung von Assembler aus C heraus

Freie Universität Berlin

- C-Seite
  - keine Aktion notwendig

- Assembler-Seite
  - AAPCS beachten, unter anderem:
    - 8 Byte Alignment für SP vor Funktionsaufruf
    - nach Funktionsaufruf können einige Register andere Inhalte haben
  - Achtung: Funktionen können aus Assembler heraus beliebig falsch aufgerufen werden!

foo.c

```
int add(int a, int b){
   return a + b;
}
```

bar.S

```
.global test
test:
   // Link-Register sichern,
   //Dummy für Alignment
   stmfd sp!, {r1, lr}

   mov r0, #42 // 1. Par. für add
   mov r1, #23 // 2. Par. für add
   bl add      // LR = PC, PC = add

/* jetzt: r0 = 42 + 23 = 65 */

   // Link-Register wieder
   // herstellen, Dummy weg
   ldmfd sp!, {r1, lr}
   bx lr // Rücksprung
```

# Assembler

- PC kann wie normales Register verwendet werden
  - bei Nutzung des PC als Input gibt es allerdings einen Versatz aufgrund der Pipeline
- ARM bietet mächtige Load/Store-Multiple-Instruktionen
  - Statt `stmfd sp!,{r0-r11}` geht auch `push {r0-r11}` (pop analog)
- Schreibweise `ldr r0, =ausdruck` ist auch nur eine Abkürzung
  - für direkte Zuweisung `mov r0, #wert` falls Ausdruck im passenden Wertebereich ist
  - für PC-relatives Laden des Wertes von Ausdruck:

```
ldr r0, ausdrucklabel
/* ... weiterer Code mit Rücksprung */
ausdrucklabel:
.word ausdruck
```

- **Achtung:** AAPCS gilt nur für normale Funktionsaufrufe. Interrupt-Handler müssen andere Regeln das Sichern von Registern und den Rücksprung beachten!

# ARMv4T Instruktionssatz

- siehe AT91RM9200 Handbuch

MOV – Move
MVN – Move Not
ADD – Add
SUB – Subtract
ADC – Add with Carry
RSB – Reverse Subtract
SBC – Subtract with Carry
RSC – Reverse Subtract with Carry
TST – Test
TEQ – Test Equivalence
CMP – Compare
CMN – Compare Negated
BIC – Bit Clear
AND – Logical AND
EOR – Logical Exclusive OR
ORR – Logical (inclusive) OR
MUL – Multiply
UMULL – Unsigned Long Multiply
SMULL – Sign Long Multiply
MLA – Multiply Accumulate
UMLAL – Unsigned Long Multiply Accumulate
SMLAL – Signed Long Multiply Accumulate
MSR – Move to Status Register
MRS – Move From Status Register
SWI – Software Interrupt

B – Branch
BL – Branch and Link
BX – Branch and Exchange
LDR – Load Word
STR – Store Word
LDM – Load Multiple
STM – Store Multiple
LDRH – Load Half Word
LDRSH – Load Signed Halfword
STRH – Store Half Word
LDRB – Load Byte
LDRSB – Load Signed Byte
STRB – Store Byte
LDRBT – Load Register Byte with Translation
STRBT – Store Register Byte with Translation
LDRT – Load Register with Translation
STRT – Store Register with Translation
SWP – Swap Word
SWPB – Swap Byte
MRC – Move From Coprocessor
MCR – Move To Coprocessor
STC – Store From Coprocessor
LDC – Load To Coprocessor
CDP – Coprocessor Data Processing