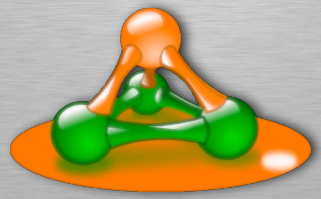
TECHNISCHE UNIVERSITÄT CHEMNITZ

Praktikum Betriebssysteme Wintersemester 2009/2010

Jan Richling*

ARM-Architektur



Prof. Matthias Werner
Professur Betriebssysteme

* Nutzung im TUC-Praktikum mit freundlicher Genehmigung

Ziele

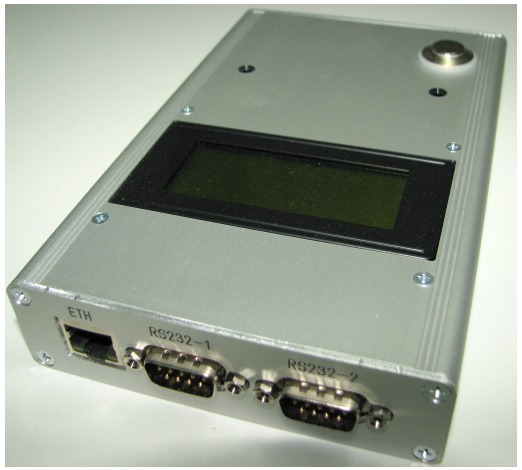
- Kennenlernen der Prozessorarchitektur der Zielplattform zwecks Entwicklung eines Betriebssystems für diese
  - Prozessormodi
  - Prozessorzustände
  - Register
  - Interrupts und Ausnahmen
- Assemblerprogrammierung auf ARM
  - Befehlssatz
  - Besonderheiten
- Speicher (am Beispiel AT91RM9200)
- Ein- und Ausgabe (am Beispiel AT91RM9200)
- Programmierung
  - Sprachen
  - Zusammenspiel von C- und Assemblerprogrammen

Zielplattform

- System
  Eingebettetes System als „System on a Chip“ (SoC)
  - SoC enthält den Prozessorkern und nahezu alle funktionalen Einheiten des Rechners
  - Darüber hinaus
    - Speicher
    - Stromversorgung
    - Schnittstellen
- Prozessor
  - Prozessorarchitektur: ARM
  - Befehlssatz: ARMv4T
  - Prozessorfamilie: ARM9
  - Prozessorkern: ARM920T
  - Prozessor: AT91RM9200

Taskit Portux MiniPC TUB

- AT91RM9200
- 16 MB Flash
- 64 MB RAM
- 180 MHz
- Ethernet 100 MBit/s
- 4 x seriell
- USB Host und Target
- Textdisplay 16x4
- Reset-Taster
- Bootloader: uboot
- Kann Linux booten



Was ist ARM?

- Eine Firma
  - Acorn RISC Machine
  - Advanced RISC Machine
  - Advanced RISC Machines, Ltd.
  - 1983-1985: Entwicklung der Acorn RISC Machine durch die Acorn, Ltd. unter Leitung von Steve Furber und Roger (Sophie) Wilson
  - Ab 1990: Weiterentwicklung und Vermarktung durch die ARM, Ltd.
- Ein Prozessordesign
- Bezeichnung von Prozessorkernen
  - Beispielweise ARM920T
- Geschäftsprinzip:
  - Verkauf von Designs in Form von Hardwarebeschreibungen
  - Keine eigene Herstellung von Hardware
  - Lizenznehmer (beispielsweise Atmel) integrieren Kerne in eigene Hardware (beispielsweise AT91RM9200)

Varianten

- T-Variante:
  Unterstützt Thumb-Modus
- M-Variante:
  Unterstützt Multiplikation zweier 32-Bit-Zahlen zu einer 64-Bit-Zahl (lange Multiplikation)
- E-Variante:
  Enhanced DSP — Instruktionen für Beschleunigung typischer DSP-Operationen
- J-Variante:
  Jazelle —bietet Soft- und Hardwarebeschleunigung für Java-Bytecode

Versionen 1-4

- Version 1:
  Nie in einem kommerziellen Produkt verwendet, 26-Bit Adressraum, nur geringe Teilmenge des aktuellen Instruktionssatzes
- Version 2:
  Erweiterte Version 1, mehr Register, Koprozessor-Unterstützung, 26-Bit Adressraum, Multiplikationsinstruktionen
- Version 3:
  32-Bit Adressraum, führt Statusregister CPSR und SPSR ein, zwei neue Prozessormodi, M-Variante
- Version 4:
  zusätzlicher (System) Prozessormodus, Thumb-Modus (T-Variante)
  - Architektur der Zielplattform
  - Alle in dieser VL vorgestellten Details gelten ab Version 4

Versionen 5-7

- Version 5:
  Erweitert Version 4 um einige Instruktionen, verbessert Zusammenarbeit zwischen Thumb- und ARM-Modus, J- und E-Variante
- Version 6:
  Instruktionen für bessere Multimedia-Unterstützung (speziell SIMD-Instruktionen), verbesserte Interrupt-Behandlung, neue Statusbits, Thumb2 optional
- Version 7:
  Thumb2 obligatorisch, neue SIMD-Instruktionen (NEON – packed SIMD), DSP-Unterstützung, Profile für
  - Anwendungen (ARMv7-A): MMU, hohe Performance bei niedrigem Energieverbrauch, optimiert für Multitasking
  - Echtzeit (ARMv7-R): Geringe Latenz, Vorhersagbarkeit, geschützter Speicher
  - Microcontroller (ARMv7-M): Geringe Transistoranzahl, Vorhersagbarkeit, tief eingebettet

Namensbildung für Implementationen

- Beginnen mit ARMv...
  - gefolgt von der Version...
  - ... und dann der Aufzählung der Varianten
  - Besonderheit: Modelle ab Version 4 unterstützen in der Regel M, so dass dies nicht mehr aufgeführt wird. Ausnahmen ohne M werden als xM gekennzeichnet
- Beispiel: ARMv5TxM
- Besonderheit: Bezeichnungen ohne “v” bezeichnen Implementationen,
- Zahl ist nicht direkt mit Architekturversion korreliert
- Beispiel:
  - Kern ARM920T ist eine Implementation der 9. Generation (ARM9) und implementiert ARM-Architektur v4T, also Version 4 mit Thumb-Modus
  - System-on-a-Chip AT91RM9200 enthält u.a. einen ARM920T-Kern

ARM: Einordnung

- Zielsysteme: Eingebettete Systeme
  - Geringer Energieverbrauch
  - Hohe Performanz (gemessen an marktüblichen Mikrocontrollern)
  - Hohe Integration (SoC)
  - Fähigkeit zur Ausführung „üblicher“ Betriebssysteme (Linux, Windows Mobile, diverse Echtzeitsysteme)
- Performanz:
  - Nicht direkt mit PC-Technik vergleichbar
  - AT91RM9200 entspricht bei Integer-Verarbeitung in etwa einem Pentium 1 mit vergleichbarer Taktfrequenz
  - AT91RM9200: 200 MIPS (Million Instructions per Second) laut Atmel
- Energieverbrauch
  - AT91RM9200 benötigt unter Vollast 25 mA bei 1,8 Volt: 45 mW
  - AT91RM9200 braucht im Standby 0,5 mA bei 1,8 Volt: 0,9 mW
  - Integrierte Peripherie kann diese Werte je nach Nutzung erhöhen LED an I/O-Port: 10 oder mehr mA
  - Üblicherweise keine besonderen Kühlmaßnahmen

Architektur-Parameter

- RISC-Design
- 32-Bit Adress- und Datenbus (max. 4 GB adressierbar)
- Load/Store-Architektur
- 3-Adress-Instruktionen mit fester Länge (32 Bit)
- 7 Prozessor-Modi
- 37 Register
- 5 grundsätzliche Adressierungsmodi mit jeweils bis zu 11 Unterformaten
- Datentypen:
  - Wort (32 Bit), an 4-Byte-Grenzen ausgerichtet
  - Halbwort (16 Bit), an 2-Byte-Grenzen ausgerichtet
  - Byte (8 Bit)
- Instruktionen
  - 54 ARM-Instruktionen: genau ein Wort breit
  - 38 Thumb-Instruktionen: genau ein Halbwort breit
  - Datenverarbeitende Operationen arbeiten alle auf Wörtern

Prozessor-Modi

Modus	Kurzform	Beschreibung
User	usr	Regulärer Benutzermodus
System	sys	Regulärer privilegierter Modus
Supervisor	svc	Modus für Systemrufe (SWI)
Abort	abt	Memory Abort
Undefined	und	Undefinierte Instruktionen
Interrupt	irq	Interrupts
Fast Interrupt	fiq	Beschleunigte Interrupts

Ausnahme-Modi

Prozessor-Modi

- Alle Modi außer User-Mode sind privilegiert
- Alle Modi bis auf User und System sind Ausnahme-Modi
- Programme laufen im User-Modus, bis Ausnahme auftritt, die im betreffenden Modus behandelt wird
- Modi haben teilweise modus-spezifische Register
- Prozessor startet im Supervisor-Modus
- Explizite Modus-Umschaltungen sind in allen privilegierten Modi möglich
- User-Modus kann nur über Software-Interrupt (SWI) aktiv verlassen werden

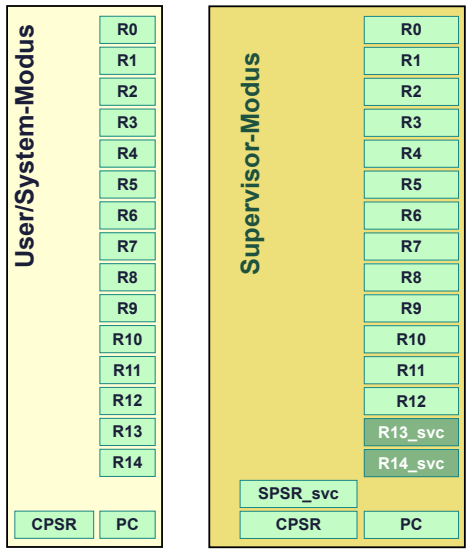
Register

- 37 Register
- Nutzersicht: R0-R15
- R0-R12 meist beliebig benutzbar
- Besondere Funktionen und Register:
  - R15: PC – Program Counter
  - R14: LR – Link Register (Ausnahme-Modus, BL-Instruktion)
  - R13: SP – Stackpointer
  - CPSR: Current Program Status Register
  - SPSR: Saved Program Status Register
- Register sind teilweise modus-spezifisch
  - Vereinfacht Moduswechsel
  - Erzeugt modus-abhängige Sicht auf Register
  - Erfordert Beachtung bei der Betriebssystementwicklung

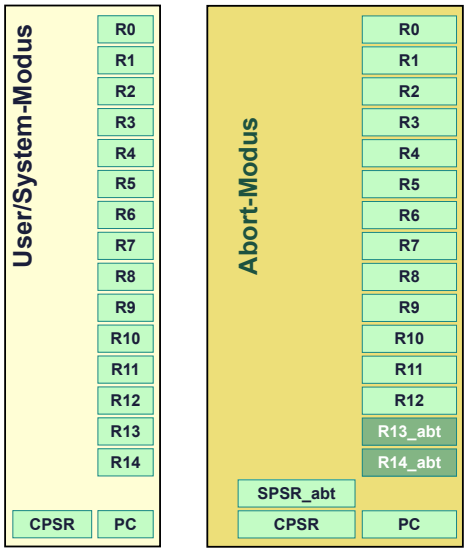
Register und Prozessor-Modi

User/System	Fast Interrupt	Interrupt	Supervisor	Undefined	Abort
R0					
R1					
R2					
R3					
R4					
R5					
R6					
R7	Fast Interrupt				
R8	R8_fiq				
R9	R9_fiq				
R10	R10_fiq				
R11	R11_fiq				
R12	R12_fiq	Interrupt	Supervisor	Undefined	Abort
R13 SP	R13_fiq	R13_irq	R13_svc	R13_undef	R13_abt
R14 LR	R14_fiq	R14_irq	R14_svc	R14_undef	R14_abt
R15 PC					
cpsr					
	spsr_fiq	spsr_irq	spsr_svc	spsr_undef	spsr_abt

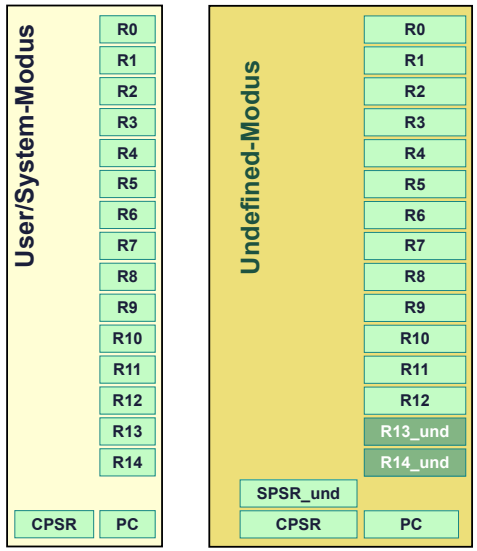
Register und Prozessor-Modi



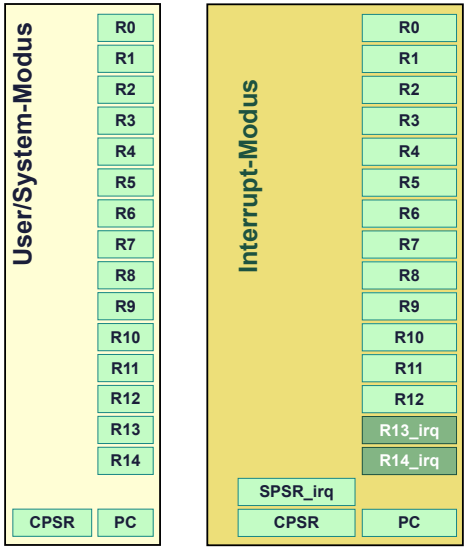
Register und Prozessor-Modi



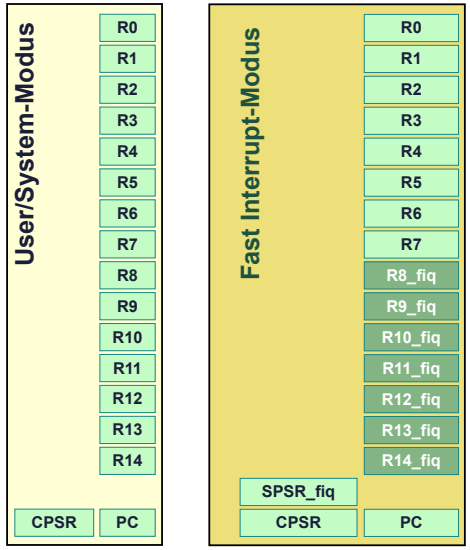
Register und Prozessor-Modi



Register und Prozessor-Modi



Register und Prozessor-Modi



Prozessor-Statusregister

- N (Negativ), Z (Null), C (Übertrag) und V (Überlauf) sind Bedingungs-Flags
- Q: Nur in E-Variante als Überlauf für erweiterte DSP-Instruktionen
- I und F: Für Interrupt- bzw. Fast Interrupt-Mode
- T: Thumb-Mode
- M kodiert den Modus der CPU
- CPSR: Current Program Status Register
  Aktueller Prozessor-Zustand
- SPSR: Saved Program Status Register
  Prozessor-Zustand vor letztem Modus-Wechsel

$M_4..M_0$	Mode
10000	usr
10001	fiq
10010	irq
10011	svc
10111	abt
11011	und
11111	sys

Ausnahmen (Exceptions)

- Wenn eine Ausnahme auftritt, geschieht Folgendes:

```
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 FIQs
CPSR[7] = 1   ; Disable IRQs
PC = exception vector address
```


Ausnahmen (Exceptions)

- Für jede Ausnahme gibt es einen Eintrag in der Interrupt-Vektor-Tabelle (IVT)
- Üblich: Interrupt-Vektor-Tabelle enthält Adressen der Behandlungsroutinen
- ACHTUNG! **Bei der ARM-Architektur enthält die Interrupt-Vektor-Tabelle ausführbaren Code!**
  (PC = exception vector address)
  - Es steht nur ein einziges Wort (32 Bit) zur Verfügung
  - Möglichkeiten
    - Sprung relativ zum Programmzähler oder Register
    - Sprung mit Zieladresse in Register
    - Sprung mit Zieladresse in Speicher relativ zu Programmzähler oder Register
    - Sofortiger Rücksprung
  - Rücksprung aus Ausnahmebehandlung erfordert Rücksetzung des Modus (SPSR ➡ CPSR) und LR ➡ PC

Ausnahmen (Exceptions): Interrupt-Vektor-Tabelle

Art der Ausnahme	Modus	Adresse IVT	Adresse HV
Reset	**Supervisor**	0x00000000	0xFFFF0000
Undefined Instruction	**Undefined**	0x00000004	0xFFFF0004
Software Interrupt	**Supervisor**	0x00000008	0xFFFF0008
Prefetch Abort	**Abort**	0x0000000C	0xFFFF000C
Data Abort	**Abort**	0x00000010	0xFFFF0010
IRQ (Interrupt)	**IRQ**	0x00000018	0xFFFF0018
FIQ (Fast IRQ)	**FIQ**	0x0000001C	0xFFFF001C

IVT: Standard-Adresse
HV: Alternative Adresse im hohen Speicherbereich

Ausnahmen (Exceptions): Prioritäten

- Ausnahmen in absteigender Priorität
  1. Reset
  2. Data Abort
  3. Fast Interrupt Request
  4. Interrupt Request
  5. Prefetch Abort
  6. Undefined Instruction (*können nicht parallel auftreten,*
  6. Software Interrupt *darum gleiche Priorität*)
- Entscheidet über Reihenfolge der Behandlung bei gleichzeitigem Auftreten
- Ausnahmen können durch Ausnahmen unterbrochen werden
  - Interrupt und Fast Interrupt können gesperrt werden
  - **Vorsicht!**
    Behandlungsroutinen müssen entsprechend programmiert werden

Befehlssätze

- ARM-Mode
  - Standard-Modus
  - Einzig möglicher Modus für Ausnahme-Behandlung
  - 54 Instruktionen
  - Für Praktikum relevant und im Folgenden vorgestellt
- Thumb-Modus
  - Höhere Code-Dichte durch 16-Bit-Instruktionen
  - 38 Basis-Instruktionen
  - Weniger mächtig als ARM
  - Wechsel jederzeit möglich: BX (Branch-and-Exchange) bei gesetztem untersten Bit der Adresse
  - Von Praktikumshardware unterstützt, aber im Praktikum nicht relevant
- Jazelle-Modus
  - Beschleunigte Ausführung von Java-Bytecode
  - Nicht weiter im Detail offengelegt
  - Von Praktikumshardware nicht unterstützt

ARM-Befehlssatz

- Alle Instruktionen sind 32 Bit lang
- Sehr strukturierte Kodierung: semantisch ähnliche Instruktionen werden auch ähnlich kodiert
- Bedingte Ausführung **aller** Instruktionen
  - Obere Bits (28-31) sind stets für die Bedingung reserviert
  - Keine Bedingung: AL (always)
- Adressierungsmodi
  - Zwei oder drei Operanden (Ziel, Quelle, optionale Quelle)
  - Ziel und optionale Quelle **immer** Register
  - Quelle: “Shifter-Operand” — kann innerhalb eines Zyklus um in der Instruktion angegebenen Wert verschoben werden
  - Je nach Typ des Befehls unterschiedliche Unterformate
- Details: Nächste Folien und ARM Architecture Reference Manual

ARM-Befehlssatz

- Allgemeine Instruktionsform:

```
<opcode>{<cond>}{S} <Rd>, <Rn>, <shifter_operand>
```

- Rd: Ziel
- Rn: optionale Quelle(n)
- Cond: Bedingung(en)
  - Wenn nicht angegeben: AL (always)
- S: S-Bit
  - Steuerung der Status-Flags
  - Steuerung der Modus-Umschaltungen

Befehlstypen

- Branch-Instruktionen
- Datenverarbeitende Instruktionen
- Multiplikations-Instruktionen
- Statusregister-Instruktionen
- Load/Store-Instruktionen
- Load/Store-Multiple-Instruktionen
- Semaphor-Instruktionen
- Instruktionen, die Ausnahmen generieren
- Koprozessor-Instruktionen

Branch-Instruktionen

```
B{<cond>} <target_address>
```

oder

```
BL{<cond>} <target_address>
```

- Berechnet PC entsprechend Zieladresse
- Zieladresse wird berechnet
- Direkte Angaben eines 32-Bit-Zieles ist **nicht** möglich
- Meist indirekt oder relativ

- L: Mit Linkregister
- Rücksprung (also Adresse nach BL) wird in Linkregister (LR, R14) gespeichert
- Kann für Unterprogrammaufrufe benutzt werden (LR retten bei Schachtelung)
- Rücksprung: `MOV PC,LR`

Datenverarbeitende Instruktionen

```
<opcode>{cond}{S} Rd,Rn,<Op2>
```

- Alle bekannten arithmetischen, logischen und Vergleichsoperationen mit Ausnahme von Multiplikation und Division
- <Op2>: Shifter-Operand oder Ausdruck

- Beispiele:

```
ADD R1,R2,R3    ; R1:=R2+R3
ADDEQ R2,R4,R5  ; Wenn Z-Flag dann R2:=R4+R5
TEQS R4,#3      ; Teste R4==3
MOV PC,R14      ; Rückkehr von Unterprogramm
MOV R0,R0       ; Leer-Instruktion (NoOP)
```


Multiplikations-Instruktionen

```
MUL{L}{cond}{S} Rd,Rm,Rs
MLA{L}{cond}{S} Rd,Rm,Rs,Rn
```

- Multiplikation und Multiplikation+Addition
  - jeweils für 32Bit x 32Bit ➡ 32Bit (obere Bits abgeschnitten) und
  - 32Bit x 32Bit ➡ 64Bit (L = Long, zusätzlich signed/unsigned)

- Beispiele:

```
MUL R1,R2,R3        ; R1:=R2*R3
MLA R1,R2,R3,R4     ; R1:=R2*R3+R4

SMULL R1,R4,R2,R3   ; {R4,R1}:=R2*R3, signed
UMLAL R1,R5,R2,R3   ; {R5,R1}:=R2*R3+{R5,R1}, unsigned
```

- Es existiert **keine** Division, muss per Bibliothek oder Koprozessor realisiert werden
  ➡ Vorsicht bei der Benutzung im Praktikum

Statusregister-Instruktionen

```
MRS Rd,CPSR
MSR CPSR,Rn
```

- Erlauben Zugriff auf Statusregister
- Umschaltungen der Modi
- Sperren/Freigeben von Interrupts

- Beispiel: Moduswechsel

```
MRS R0,CPSR          ; Kopiere CPSR nach R0
BIC R0,R0,#0x1F      ; Modus-Bits löschen
ORR R0,R0,#new_mode  ; Neuen Modus setzen
MSR CPSR,R0          ; Zurückschreiben
```


Load/Store-Instruktionen

```
<LDR|STR>{cond}{B}{T} Rd,<Address>
```

- Adresse wird als Ausdruck berechnet
- Sehr viele Möglichkeiten: Siehe Handbuch
- Meist relativ, indirekt oder über Register, da keine 32-Bit-Werte möglich sind
- Unmittelbar-Operanden: Relativ über PC
- B: Byteweise, sonst wortweise

- Beispiele:

```
LDR R1,[R2,#16]  ; Lade R1 mit Inhalt von
                 ; Adresse (R2+16)
STR R1,[R2,R4]!  ; Speichere R1 auf Adresse
                 ; (R2+R4) und schreibe
                 ; Adresse nach R2
STR R1,[R2],R4   ; Speichere R1 nach Adresse in R2
                 ; und schreibe R2+R4 zurück
                 ; nach R2
```


Load/Store-Multiple-Instruktionen

```
<LDM|STM>{cond}<FD|ED|FA|EA|IA|IB|DA|DB> Rn{!},<Rlist>{^}
```

- Erlaubt das Laden von Registern mit dem Inhalt **aufeinanderfolgender** Speicheradressen bzw. das Speichern von Registern an aufeinanderfolgende Speicheradressen
- Mehrere Varianten:
  - IA, IB, DA, DB:
    I: Increment, D: Decrement, B: Before, A: After
  - FD, ED, FA, EA: Für Stackoperationen (nächste Folie)
- Beispiel: **`STMDA R13!, {R0 - R12, LR}`**
  - STDMA — Store Multiply Decrement After
  - Indexregister wird nach jedem Schritt dekrementiert
  - R13 ist Indexregister mit der Basisspeicheradresse
  - ! besagt, dass R13 nach jedem Schritt aktualisiert wird
  - R0-R12 und LR sind die Register, die in den Speicher kopiert werden sollen
  - Ablauf: R0 nach Adresse, die in R13 steht, R13 um 4 dekrementieren, R1 an diese Adresse, usw.

Load/Store-Multiple-Instruktionen

- Ist nützlich beim Umgang mit dem Stack:
  Kann einen ganzen Satz von Daten vom Stack in die Register kopieren (oder umgekehrt) mit Aktualisierung des SP
- Richtung des Stapel-„Wachstums“ wird durch LDM/STM-Befehl festgelegt
- Es gehören immer „gleichlautende“ Befehle zusammen:
  Auf Stack legen mit STMFD erfordert Lesen mit LDMFD
- Möglichkeiten: FD, ED, FA, EA
  - Full: Erst Zeiger ändern, dann schreiben
  - Empty: Erst schreiben, dann Zeiger ändern
  - Ascending: Aufsteigender Stapel, STM geht nach oben, LDM nach unten
  - Descending: Absteigender Stapel, STM geht nach unten, LDM nach oben

Atomare Swap (für Semaphore)

```
<SWP>{cond}{B} Rd,Rm,[Rn]
```

- SWP (Swap) und SWPB (Swap Byte) erlauben atomare Lese/Schreibe-Operationen
- Beispiel 1:
  - `SWP R1, R2, [R3]`
  - R3 enthält Speicheradresse
  - SWP lädt R1 mit dieser Speicheradresse und schreibt R2 an diese Adresse
- Beispiel 2:
  - `SWP R1, R1, [R2]`
  - R2 enthält Speicheradresse
  - SWP tauscht Inhalt von R1 und der Speicheradresse
- Atomizität stellt sicher, dass es keine Unterbrechung zwischen den Teilen der Operation geben kann

Trap

```
SWI{cond}
```

- Löst einen Software-Interrupt (Trap) aus
  - Wechselt in Supervisor-Mode
  - PC wird auf 0x8 gesetzt
  - LR speichert Rücksprung-Adresse
- Kann zur Realisierung von Systemrufen benutzt werden
  - Register 0-3 können zur Unterscheidung verschiedener Funktionen benutzt werden

Koprozessor-Instruktionen

- ARM-Design ermöglicht einfache Erweiterungen durch die Koprozessor-Architektur
  - Bis zu 16 Koprozessoren an einem ARM-Kern
  - ARM ist für Flußsteuerung zuständig
  - Lagert Berechnungen aus
  - Kommunikation über Bus und Ausnahmen
    - Wenn kein Koprozessor reagiert, gibt es eine undefined-Ausnahme
    - Behandlung dieser Ausnahmen ermöglicht Software-Emulation
- Koprozessoren:
  - Wie ARM load/store
  - Bis zu 16 Register
  - Beliebige interne Verarbeitungsbreite
- Beispiele:
  - Fließkomma-Berechnungen (FPU, auf Zielplattform nicht vorhanden)
  - System Control Processor (CP15 auf Zielplattform, steuert u.a. MMU, relevant für Speicherschutz-Aufgabe)
  - Debug Communication Channel (CP14 auf Zielplattform)

Koprozessor-Instruktionen

```
CDP{cond} p#,<expression1>,cd,cn,cm{,<expression2>}
<LDC|STC>{cond}{L} p#,cd,<Address>
<MCR|MRC>{cond} p#,<expression1>,Rd,cn,cm{,<expression2>}
```

- P#: Nummer des Koprozessors
- Ausdrücke werden zu einer Konstante berechnet und weitergegeben
- cd, cn und cm sind Koprozessor-Befehle und –Register
- CDP: Koprozessor-Datenoperationen
- LDC/STC: Koprozessor-Datentransfer
- MCR/MRC: Koprozessor-Registertransfer
- Beispiele:

```
CDP p1,10,c1,c2,c3    ; CP1 führt Op. 10 auf c2 und c3
                      ; durch und legt das Ergebnis nach c1
LDC p1,c2,table       ; Lade c2 von CP1 aus Tabelle relativ zu PC
MCR p6,0,R4,c5,c6     ; CP6 führt Operation 0 auf R4 durch und
                      ; schreibt Ergebnis nach c4
MRC p2,5,R3,c5,c6     ; CP2 führt Operation 5 auf c5 und c6 durch
                      ; und schreibt Ergebnis nach R3
```

Besonderheit: Bedingte Ausführung

- Bedingte Ausführung aller Instruktionen spart Sprünge ein
- Verbessert Performance (Sprungvorhersage, Pipeline)
- Kompakterer Code
- Beispiel:

```
if (a == 0)
{
  b = 23;
}
else
{
  b = 42;
  c = a;
}
```


Besonderheit: Bedingte Ausführung

- In IA32-Code (ohne bedingte Ausführung aller Befehle)

```
TEST EAX, EAX
JNE _else
MOV EBX, 23
JMP _done
_else: MOV EBX, 42
MOV ECX, EAX
_done:
```


Besonderheit: Bedingte Ausführung

- In ARM-Code (ohne bedingte Ausführung aller Befehle)

```
TEQ R0, #0
BNE else
MOV R1, #23
B done
else: MOV R1, #42
MOV R2, R0
done:
```

- In ARM-Code (mit bedingter Ausführung aller Befehle)

```
TEQ R0, #0
MOVEQ R1, #23
MOVNE R1, #42
MOVNE R2, R0
```


Besonderheit: Shifter-Operand

- Innerhalb einer Operation kann ein Quelloperand verschoben oder rotiert werden:

```
MOV R1, R0, LSL #3        ; R1 := R0 * 8
ADD R3, R2, R1, LSR #2    ; R3 := R2 + R1 / 4
MOV R3, R2, ROR R1        ; R2 um R1 nach
                          ; rechts rotieren
                          ; und Wert in R3
                          ; laden
```

Besonderheit: S-Bit

- Viele ARM-Befehle haben ein Feld für das S-Bit
- Bedeutung variiert mit Kontext
- Oft:
  - Wenn PC nicht als Ziel angegeben
    - Setze Bedingungsflags
    - Benutze User-Register statt modus-spezifischen Registern
  - Wenn PC als Ziel angegeben
    - Lade nach Ausführung CPSR mit Inhalt von SPSR
- Genaue Interpretation ist von Befehl und Zustand abhängig
- Beispiel: **`LDM/STM`** mit gesetztem S-Bit
  - LDM mit PC in Transferliste: SPSR_mode ➡ CPSR
  - STM mit PC in Transferliste: User- statt Modus-Register
  - R15 nicht in Transferliste: User- statt Modus-Register
- Verwendung: Rückkehr aus Ausnahmebehandlungen:

```
MOVS  PC,LR
```


Thumb-Befehlssatz

- 16 Bit Instruktionslänge
- 38 Basis-Instruktionen
- Jederzeit Zugriff auf 8 Register (R0-R7)
- Einzelne Instruktionen bieten auch Zugriff auf PC (R15), LR (R14), SP (R13) oder High Registers (R8-R15)
- Wechsel zur Laufzeit (mit BX, BLX, SPSR,LDR/LDM)
- Ausnahmen werden **immer im ARM-Modus** behandelt — in jedem ARM-System existiert damit gemischter Code

- Von Praktikumshardware unterstützt
- Im Praktikum nicht weiter betrachtet
- Wer mag, darf gerne Teile des OS im Thumb-Modus implementieren

Thumb-Befehlssatz

- Branch-Instruktionen
- Datenverarbeitende Instruktionen
- **Keine** Multiplikations-Instruktionen
- **Keine** Statusregister-Instruktionen
- Load/Store-Instruktionen
- Load/Store-Multiple-Instruktionen
- **Keine** Semaphor-Instruktionen
- Instruktionen, die Exceptions generieren
- **Keine** Koprozessor-Instruktionen

Thumb-Befehlssatz

- Instruktionsform:

```
<opcode> <Rd>, <Rn>, <3rd_operand>
```

- 3. Operand: Register oder kurzer Immediate-Wert

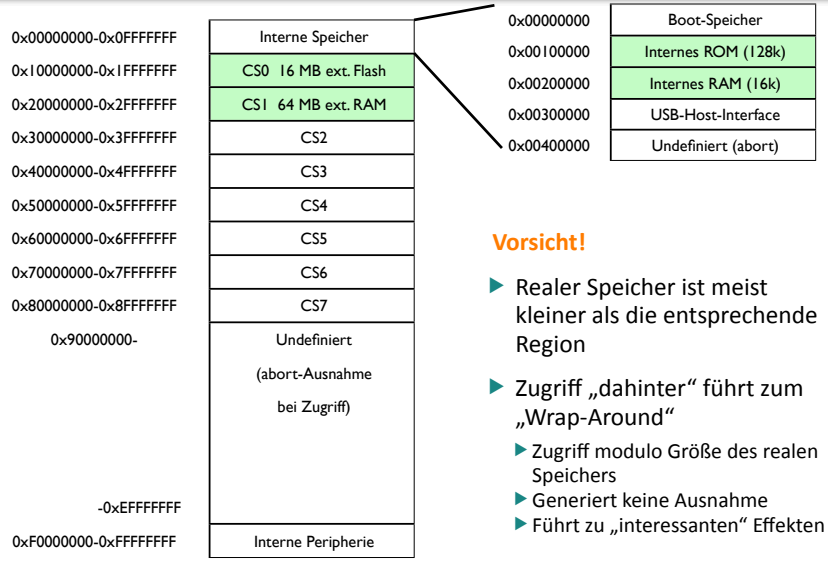
Speicher

- ARM-Kern stellt 32-Bit-Busse für Adressen und Daten bereit
  - Kann bei byteweiser Adressierung maximal 4 GB adressieren
- Kern kann Speicherverwaltungseinheit (Memory Management Unit, MMU) enthalten
  - Optional, nicht bei allen Kernen
  - ARM920T hat MMU
- Verwendung der Adressleitungen des Kerns hängt von Implementation der Peripherie ab
- Praktikum:
  AT91RM9200 in der Konfiguration der Zielplattform
  - Interne Speicherbereiche (Teil des AT91RM9200)
  - Externe Speicherbereiche (durch Taskit hinzugefügt)
  - Auf Speicher abgebildete Ein/Ausgabe (memory-mapped I/O)

Speicher der Zielplattform

- Gliedert die 32 Adressleitungen auf in
  - 4 Bit für die Auswahl einer „Chip-select Region“
  - 28 Bit für die Adressierung innerhalb der Chip-select-Region
- Idee dabei:
  - Ermöglicht auf einfache Weise die Integration sehr unterschiedlicher Speicher
  - Auswahl der Chip-select-Region generiert „Chip-select“-Signal für jeweils eine Komponente (verschiedene Speicher, I/O)
- Theoretisch: 16 Regionen (2^4=16)
- Praktisch:
  - 8 Regionen (CS0 bis CS7) von jeweils 256 MB (2^{28} Byte)
  - 2 spezielle Bereiche
    - Interne Speicher
    - Ein/Ausgabe
  - Zugriff auf übrige Regionen führt zur Abort-Ausnahme

Speicher der Zielplattform



Vorsicht!

- Realer Speicher ist meist kleiner als die entsprechende Region
- Zugriff „dahinter“ führt zum „Wrap-Around“
  - Zugriff modulo Größe des realen Speichers
  - Generiert keine Ausnahme
  - Führt zu „interessanten“ Effekten

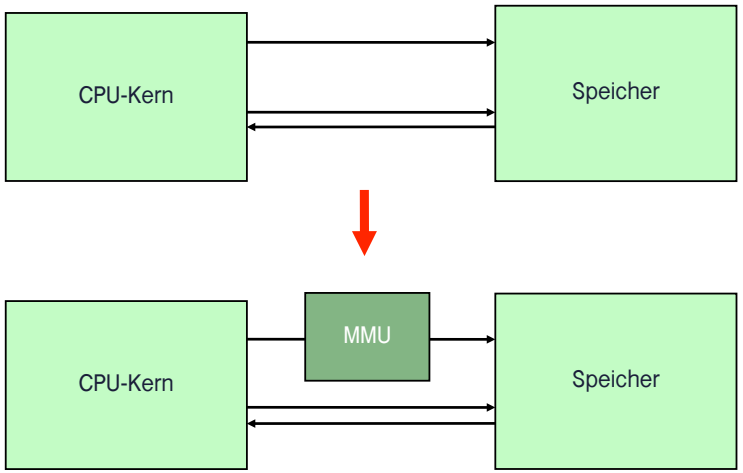
Boot-Speicher

- ARM-Kern startet mit Ausführung auf Adresse 0x00000000
- Zur Erinnerung: Dort liegt der Reset-Vektor
- Problem: An dieser Stelle darf beim Anschalten kein flüchtiger Speicher liegen!
- AT91RM9200 kann in Abhängigkeit von einer externen Leitung (BMS) und dem „Remap“-Kommando verschiedene Speicher dorthin abbilden:
  - Vor Remap, BMS==0: CS0 (externer Flash)
  - Vor Remap, BMS==1: Internes ROM
  - Nach Remap: Internes RAM
- Zielplattform: BMS==0
  Bootet damit aus dem Flash den Bootloader U-Boot
- Um die Interrupt-Vektor-Tabellen ändern zu können, **muss** ein Remap durchgeführt werden
  - Tabellen liegen dann im internen RAM ab 0x00200000
  - Können auch unter 0x00000000 zugegriffen werden
  - Alternative: Umschalten auf „hohe“ Tabellen ab 0xFFFF0000

Memory Management Unit

- ARM920T enthält Speicherverwaltungseinheit (Memory Management Unit, MMU)
  - Realisiert Speichervirtualisierung
  - Abbildung virtueller Adressen auf physikalische
  - Abbildung wird durch Seitentabellen definiert
  - Transparent für den Kern
  - Realisiert zusätzlich Speicherschutz (Memory Protection Unit, MPU)
  - Wird über Koprozessor 15 gesteuert
- Kann physikalisches Speicherlayout vollständig verbergen
- **Vorsicht!**
  - Für gezieltem Zugriff auf bestimmte Adressen (z.B. Interrupt-Vektor-Tabelle, Ein/Ausgabe) muss für entsprechende Abbildung gesorgt werden
- Wird durch Betriebssystem gesteuert

Memory Management Unit



Manipulation der von der CPU angesprochenen Adressen

Ein/Ausgabe

- ARM-Kern kommuniziert im AT91RM9200 mit Umgebung (Rest des SoC und dessen Umfeld) über
  - Daten/Adressbus
  - Zwei Interrupt-Leitungen (IRQ und FIQ)
  - Debug-Schnittstellen (gesteuert von Koprozessor 14)
- **Keine** direkten Ein/Ausgabe-Leitungen
- Auf Speicher abgebildete Ein/Ausgabe (memory-mapped I/O)
  - Gerät hat Register für
    - Konfiguration
    - Datenaustausch
  - Register werden in den Adressraum des Kerns **eingeblendet** (0xFFFA0000 bis 0xFFFFFFFF)
  - Zugriff führt zu entsprechenden Reaktionen des Gerätes
  - Oft kann das Gerät einen Interrupt auslösen
    - Erspart „Polling“
    - Behandlungsroutine fragt das Gerät ab und führt angeforderte Operation durch

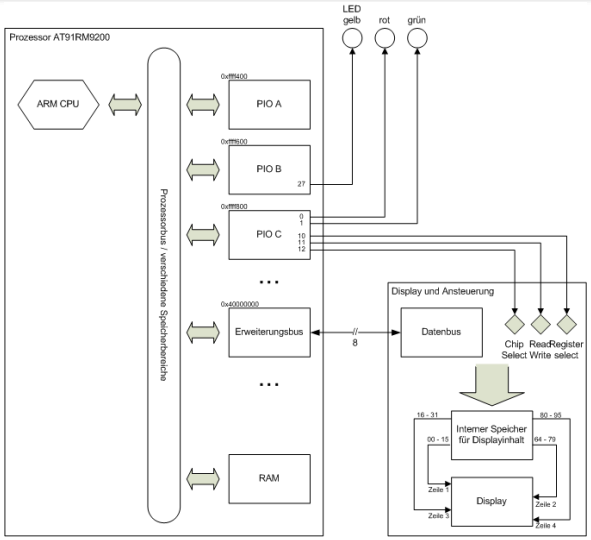
Ein/Ausgabegeräte des AT91RM9200 (Auswahl)

- Parallel I/O (PIO)
  - Direkt steuerbare Ein/Ausgabeleitungen
  - Beispiel: Ansteuerung der LEDs (siehe Beispielprogramm)
- System-Zeitgeber
  - Kann periodische Interrupts generieren
- Interrupt-Controller
  - Schaltet Interrupt-Leitungen der Peripheriegeräte auf IRQ und FIQ des Kerns
  - Interrupt-Quellen einzeln konfigurierbar
- Serielle Schnittstellen
- Ethernet
  - Im Praktikum benutzt für Download des Codes
- USB
- ...
- Details: Handbuch des AT91RM9200

Spezialfall Display

- Einzige extern hinzugefügte Ausgabeeinheit der Zielplattform
- Ist an PIO und Speicherbus angeschlossen
  - PIO: Steuerung (Lese/Schreib-Umschaltung, Signalisierung, Chip-Select, …)
  - Speicherbus: Datentransfer
    - Ist an CS3 (0x40000000-0x4FFFFFFF) angeschlossen
    - Benötigt nur eine Adresse
    - Wrap-Around: Jede Adresse aus 0x40000000-0x4FFFFFFF kann genutzt werden
- Ansteuerung muss die geeignete Sequenz aus Steuersignalen und gesendeten Datenwörtern erzeugen
  - Spätere Aufgabe

Spezialfall Display: Anbindung



Programmierung

- GNU-Tools für übliche Programmiersprachen
  - gcc als Crosscompiler für ARM
  - cross-binutils für Linken
  - cross-as als Assemblierer für ARM
- Im Praktikum
  - C für alles, was in C machbar ist
  - Assembler für maschinennahe Operationen (z.B. Behandlung von Ausnahmen)
  - Ganz ohne Assembler **funktioniert es nicht**!
- Herausforderung
  - Kombination von Assembler- und C-Teilen
  - Sinnvolle Aufgabenteilung
  - Empfehlung: Möglichst viel in C!

Kombination von C und Assembler

- Inline-Assembler
  - Einzelne Instruktionen direkt in C-Code einbetten
  - Beispiel Moduswechsel:

```
printf("Wechsle Modus...\n");
asm("MRS r1,cpsr");
asm("BIC r1,r1, #0x80");
asm("MSR cpsr_c,r1");
printf("Modus gewechselt!\n");
```

  - **Vorsicht!**
    Benutzung von Registern hat Einfluss auf C-Programm!
- Assembler-Objekt-Files mit C-Objekt-Files linken
  - Trennung des Codes
  - Zusammenspiel über Einsprungadressen und Parameterübergabe (nächste Folie)
  - Linken

Übergabe von Parametern

- Üblicher Weg:
  - Parameter einer Funktion werden auf den Stack gelegt
  - Aufgerufene Funktion liest sie
  - Legt am Ende das Ergebnis auf Stack
  - Aufrufer liest es
- ARM nach ARM Procedure Call Standard (APCS):
  - Erste vier INT-Parameter werden über R0 bis R3 übergeben
  - 64-Bit-Werte paarweise mit R0/R1 und R2/R3
  - Rückgabe: R0 bzw. R0/R1 bei 64 Bit
  - Übrige Parameter: Stack
- ARM-GCC:
  - Folgt APCS, legt die Parameter dann aber auf den Stack
- **Vorsicht!**
  - Übergabe über Register ist nicht wiedereintrittsfähig, Retten der Parameter auf geeignete Weise (z.B. Stack)
  - Benutzung von Registern hat wieder Einfluss auf C-Programm!

Übergabe von Parametern

- Beispielcode LED-Ansteuerung
- Assemblerroutine initialisiert LED-Port
  - Parameter: Basisadresse des PIO-Controllers
  - Rückgabe: 1
  - Benutzung des Stack zur Rettung verwendeter Register
- C-Routine ruft Assemblerroutine auf
  - Übergibt Basisadresse für Controller
  - Testet Rückgabewert

Assemblerseite

```
led_init:
   stmfd sp!, {r2}              ;save used registers
   mov r2, #YELLOW_LED_REGISTER ;set r1 with yellow
   str r2, [r0]                 ;enable I/O for yellow
                                ;led

   add r0, #PIOB_OER_OFFSET
   str r2, [r0]                 ;enable output for
                                ;yellow led

   ldmfd sp!, {r2}              ;restore used registers
   mov r0, #1                   ;set return value TRUE
   mov pc, lr                   ;return
```


C-Seite

```
if(led_init((PIOB_REGISTER_ADDRESS)PIOB_BASE)==1)
   {
      // do something
   }
```

Literatur

- Bücher
  - ARM System Architecture — Steve Furber
  - ARM System Developer‘s Guide – Andrew Sloss, Dominic Symes, Chris Wright
- Online-Quellen (siehe Links auf Webseite)
  - ARM Architecture Reference Manual
  - ARM Assembly Language Programming
  - Referenzhandbuch zum ARM920T
  - Referenzhandbuch zum AT91RM9200

Wie geht es weiter?

- Anschließend: Einführung Pool + Vorführung Tool-Chain
- Nächste Woche: Besprechung Aufgabe 1
- Bis dahin:
  - Jeder baut Beispielcode nach und führt ihn aus