# Kapitel 5
# Andere Mikroprozessor-Architekturen

# 5.1 Überblick

Bei **allgemeinen Rechnern** (*Computer*) sind folgende **Mikroprozessoren** verbreitet:

| *CPU-Typ* | *Hersteller* | *Art/Datenwortbreite* | *Bemerkung* |
|---|---|---|---|
| • **Arbeitsplatzrechner** (PC, Workstation) | | | |
| 80x86 | Intel, AMD | CISC, 32 (64) bit | Markennamen Pentium, Core, Athlon, Opteron, … |
| Power PC | IBM, Freescale | RISC, 32bit | Bis 2005 in Apple-Computern |
| • **Server** (File-Server, Web-Server, Datenbank-Server) | | | |
| 80x86 | Siehe oben | | |
| Itanium | Intel (HP) | RISC, 64bit | |
| Sparc | Sun | RISC, 64bit | Gefertigt von Fujitsu u.a. |
| Power | IBM | RISC, 64bit | Aufwärtskompatibel zu Power PC |
| • **Großrechner** für kommerzielle Aufgaben (Mainframe) und technische Berechnungen (Vektorrechner, Number Cruncher) | | | |
| Diverse | IBM, Cray, SGI, NEC, Fujitsu | | Eigenentwicklungen oder Cluster mit mehreren Tausend Standard-Mikroprozessoren (z.B. IBM Blue Gene mit Power PC, Cray XT mit Opteron, SGI Altix mit Itanium) |

Bei **mobilen Computern** dominieren folgende **Mikroprozessoren**:

| | | | |
|---|---|---|---|
| • **Spielekonsolen** | | | |
| Power | IBM | RISC, 64bit | Microsoft XBOX 2.Generation, Nintendo Wii/Game Cube |
| Cell | IBM (Sony, Toshiba) | RISC, 64bit Multi-Kern-CPU | Sony Playstation 3 (enthält zusätzliche Power PC CPU für Verwaltungsaufgaben) |
| • **Taschencomputer und Mobiltelefone** (Handheld, PDA) | | | |
| ARM 9, ARM 11 | Diverse | RISC, 32bit | Intel/Marvell XScale, Texas Instruments OMAP u.a. mit DSP- und Java-Erweiterungen und/oder Signalprozessor Texas Instruments TMS320… oder Analog Devices Shark/Blackfin |

# 5.1 Überblick

Bei **eingebetteten Systemen** (*Embedded Systems*) dominieren folgende **Mikrocontroller**:

| *CPU-Typ* | *Hersteller* | *Art/Datenwortbreite* | *Bemerkung* |
|---|---|---|---|
| 8051 | (Intel), Diverse | CISC, 8bit | Weiterentwicklung von sehr vielen Herstellern, z.B. Phillips/NXP, Siemens/Infineon u.a. |
| 680x | Freescale/Motorola | CISC, 8bit | Familien 6805, 6808, 6809 |
| 681x | | CISC, 16bit | Familien 6811, 6812, 6816 |
| 68xxx | | CISC, 32bit | Familie 68000 – 68060, 68332, ColdFire, … |
| MPC55xx | | RISC, 32bit | Embedded Power PC MPC555, 5556, … |
| PIC 1x<br>PIC 24 | Microchip | RISC, 8bit<br>RISC, 16bit | Familien 12, 14, 16, 18 |
| AVR 8<br>AVR 32 | Atmel | RISC, 8bit<br>RISC, 32bit | ATtiny, ATmega |
| C16x<br>TriCore TC1xxx | Infineon | RISC, 16bit<br>RISC, 32bit | Nur in Europa stark verbreitet |
| 78Kxx<br>V850 | NEC | CISC, 8/16bit<br>RISC, 32bit | |
| R8C<br>M16C<br>R32C<br>SuperH | Renesas | Sehr viele Produktfamilien vom 4bit bis zum 32bit Mikroprozessor/Mikrocontroller, da Renesas als Zusammenschluss mehrere Hersteller (Hitachi, Mitsubishi) entstanden ist. | |
| ARM 7<br>ARM Cortex | Diverse | RISC, 32bit | Lizenziert an sehr viele Halbleiterhersteller, u.a. Freescale, Atmel, Philips, STM, … |

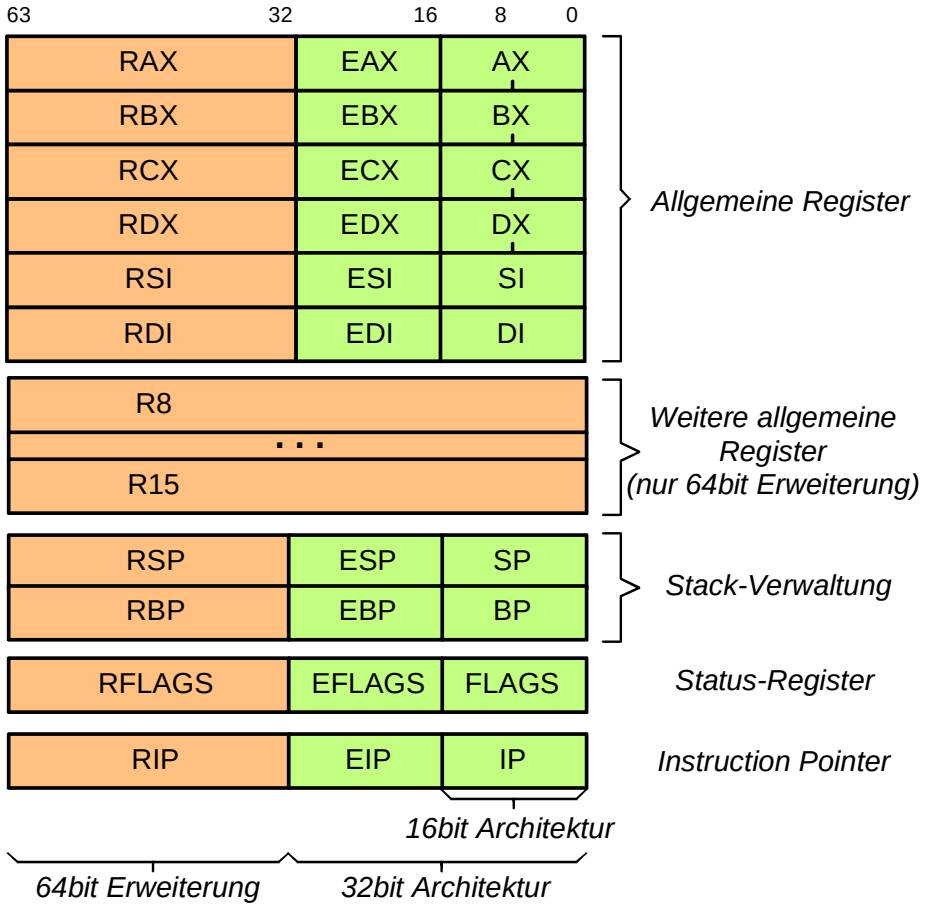
Sehr dynamischer Mark, z.T. sehr „alte“ Architekturen (8051, 68xx aus den 70er Jahren), große Leistungsunterschiede zwischen den CPUs, stärkstes Wachstum bei ARM7.

## 5.2 80x86-Architektur

(siehe auch http://developer.intel.com)

- **Programmiermodell** für Anwendungsprogramme

Registersatz




Gleitkommaregister (FPU, SSE) nicht dargestellt.

### Historische Wurzeln

- 8bit Architektur 8080, 8085 (1974)

Seit 1979 aufwärtskompatibel weiterentwickelt

- 16bit Architektur 8086, 80186, 80286
- 32bit Architektur 80386, 80486, Pentium
- 64bit Architektur AMD64, Core EM64
  64bit Erweiterung von den Standardbetriebssystemen Windows XP/Vista und Linux derzeit noch nicht genutzt

### Mehrere Betriebsmodi

- Real Mode 16bit DOS (IBM PC 1981)
- Protected Mode 32bit Windows (Win32)
- Kompatibilitätsmodus 16bit/32bit (Win16)
- 64bit Modus
- Kompatibilitätsmodus 32/64bit (Win64)

### Charakteristische Eigenschaften

- Kleiner Registersatz (nur 6 allgemeine 32bit Register)
- **CISC**-Befehlssatz mit >> 100 Befehlen
- **Von-Neumann**-Speichermodell
- **Little Endian**, Byte-adressierbar
- Adress- und Datenwortbreite 32bit (64bit)
- **2 Adress-Befehle**, 1 Register-Operand und 1 Register- oder Speicher-Operand

Beispiel für 80x86-Maschinenbefehle:

```
.data
myVar DD 01234567h, 89ABCDEFh
```

Definition eines Arrays mit zwei 32bit Elementen

```
.code
MOV ESI, 4
MOV EAX, 1
ADD EAX, myVar[ESI]
```

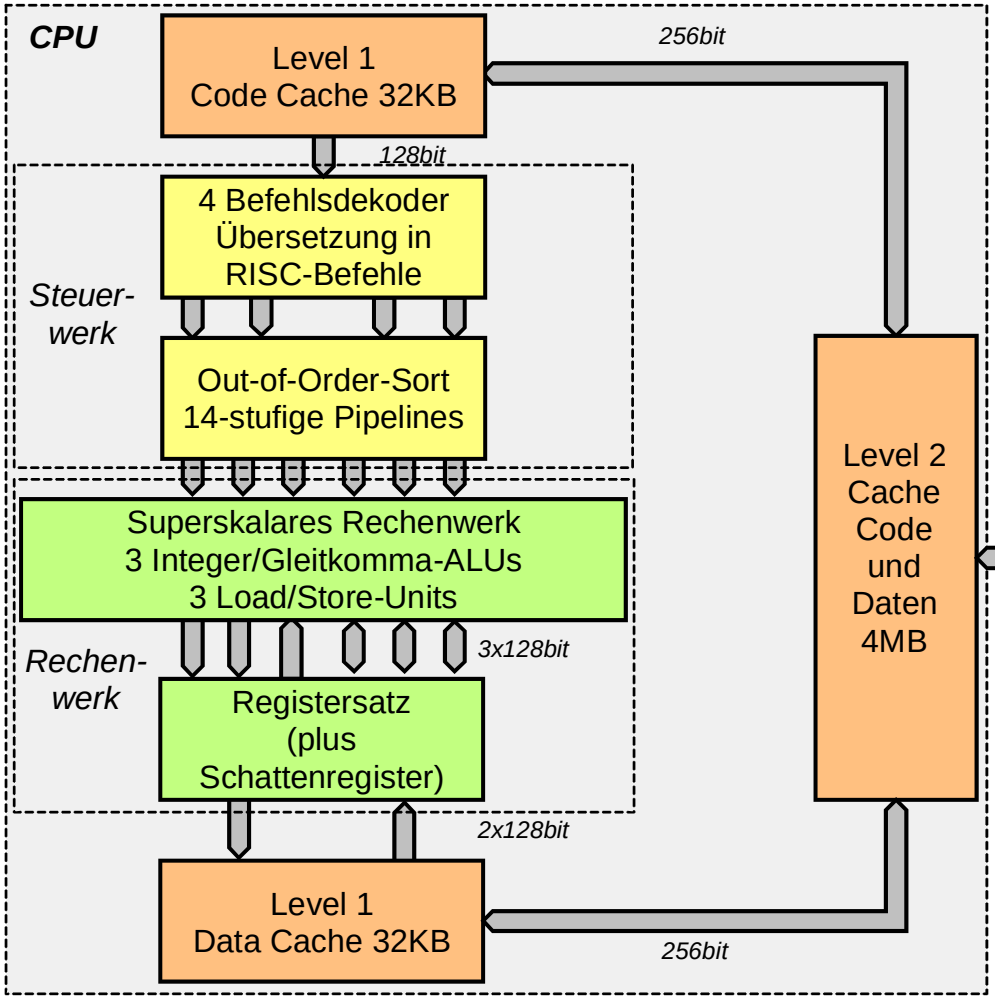
**ESI = 4**
**EAX = 1**
**EAX = EAX + myVar[ESI] = 1 + 89ABCDEFh**

Mögliche Adressierungsarten
- Registeradressierung
- Unmittelbare Adressierung
- Direkte, Register-Indirekte und Indizierte Speicheradressierung
- Single Instruction Multiple Data (SIMD) Befehle mit bis zu 4 x 2 32bit Operanden
- Gleitkommabefehle mit 32bit, 64bit und 80bit Gleitkommaoperanden

Orthogonaler Befehlssatz
- Jedes Register kann als Quelle oder Ziel bei (fast) jedem Befehl verwendet werden

| *Auffällige Unterschiede zum HCS12* | *Intel* | *Motorola/Freescale* |
|---|---|---|
| • Speicherreihenfolge | Little Endian | Big Endian |
| • Operandenreihenfolge | Ziel – Quelle<br>z.B. **MOV EBX,EAX  EBX ← EAX** | Quelle – Ziel<br>**z.B. TFR A,B  A → B** |
| • Konstanten | **MOV EAX, 1** | **LDAA #1** |
| • Hexadezimale Werte | **01234567h** | **$01234567** |

## Mikroarchitektur aktueller 80x86-CPUs (Core 2 Stand 9/07)



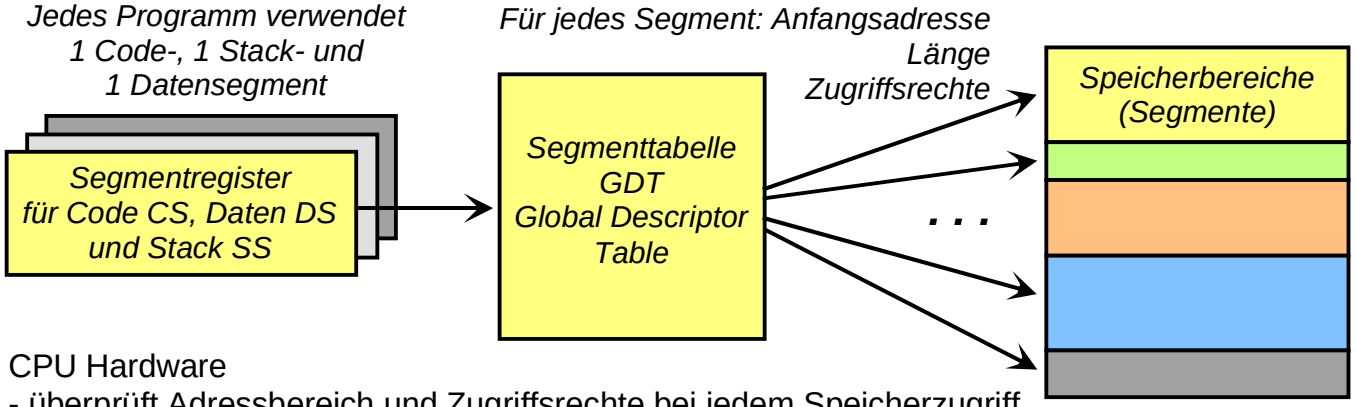

*Nur Datenpfade Adress- und Steuersignale nicht dargestellt*

Die tatsächliche **Mikroarchitektur** ist wesentlich komplexer als das 30 Jahre alte Programmiermodell:

- **Intern**er Speicher in **Harvard**-Struktur, externer Speicher in Von-Neumann-Struktur
- 80x86-CISC-Befehlssatz wird **intern** in 4 Befehlsdekodern in **RISC-Befehlssatz** übersetzt
- **Superskalares Rechenwerk** (Multiple Issue mit max. 6 RISC-Befehlen parallel)
- **Mittellange Pipeline** mit ca. 14 Stufen
- **Out-of-Order**-Ausführung mit umfangreichem Satz von **Schattenregistern**

## CPU-Sicherheitsmechanismen zur Unterstützung von Betriebssystemen

- Privilegstufen
  - Jedem Programm wird eine von zwei Privilegstufen zugeordnet: *Kernel Mode – User Mode. Die* Zuordnung erfolgt über die sogenannten *Segmentregister* für Code, Daten sowie Stack und die *Segmenttabelle*
  - Betriebssystem läuft im *Kernel Mode*, Anwendungsprogramme laufen im *User Mode*
  - *User Mode* Programme haben keinen Zugriff auf *Kernel Mode* Daten
  - *User Mode* Programme können *Kernel Mode* Funktionen nur über definierte Adressen (*Call Gates*) aufrufen
- Privilegierte Befehle
  - Kritische Befehle wie das Sperren/Freigeben von Interrupts, Ein-/Ausgabebefehle für den Zugriff auf Hardware-Register und das Lesen/Schreiben von Registern für den Speicherzugriffsschutz sind nur im *Kernel Mode* möglich.
- Speicherverwaltung und -zugriffsschutz →Memory Protection MPU und Memory Management MMU



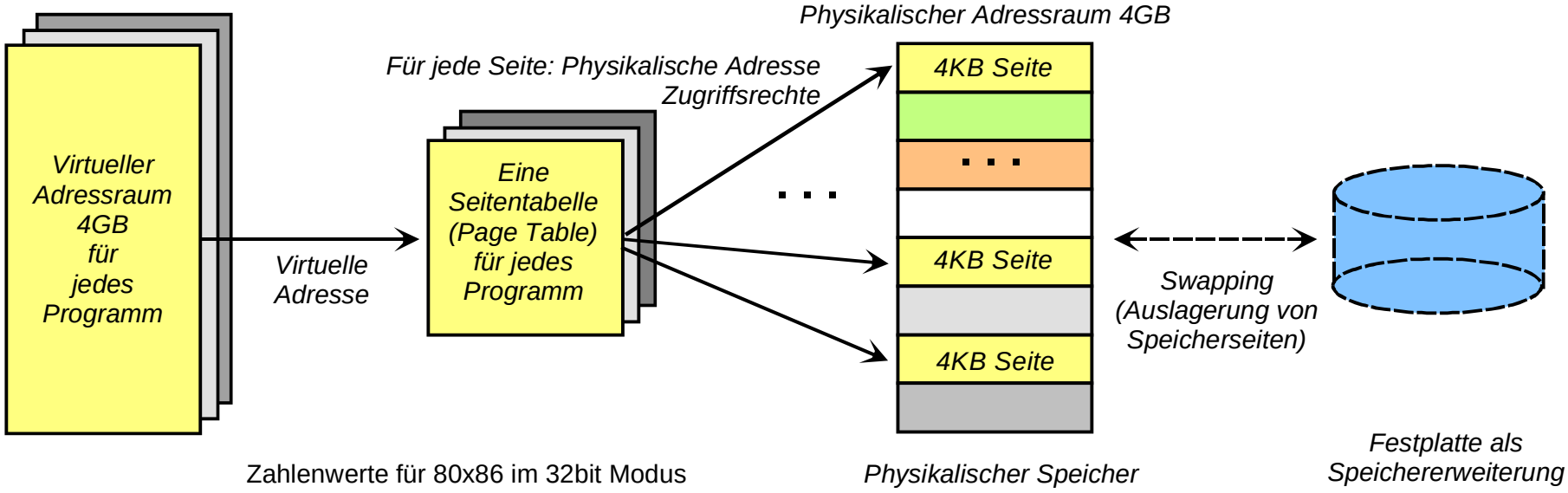

CPU Hardware
- überprüft Adressbereich und Zugriffsrechte bei jedem Speicherzugriff
- bricht Programm ab (Exception Interrupt), falls Zugriff unzulässig

Memory Protection durch **Segmentierung**

- Einteilung des Speichers in Bereiche variabler Länge (Segmente)
- Zugriffsrechte für jedes Segment:
  Code – Daten
  Kernel – User Mode
  Read–Write–Execute

Memory Management durch **Seitenverwaltung (Paging)**

- Jedes Programm erhält virtuell den gesamten 4GB Adressraum für sich allein
- Einteilung des Speichers in Bereiche fester Größe, z.B. 4KB (*Seiten, Pages*)
- Abbildung der virtuellen Seiten auf reale physikalische Speicherseiten über Seitentabellen
- Jedes Programm erhält eine eigene Seitentabelle mit seinen individuellen Seiteneinträgen, d.h. ein Programm sieht nur seine eigenen Speicherbereiche, nicht die Seiten anderer Programme
- Falls der physikalische Speicher nicht ausreicht, werden Seiten, die gerade nicht gebraucht werden, auf die Festplatte ausgelagert (*Swapping*)
- Speicherseiten werden mit Zugriffsrechten versehen. Beim Zugriff auf eine Speicherseite, die nicht in die Seitentabelle eingetragen ist oder andere Zugriffsrechte erfordert, wird ein *Exception Interrupt* ausgelöst. Die ISR des Betriebssystems entscheidet dann, ob das Programm abgebrochen, eine neue Seite eingerichtet (z.B. bei automatischer Vergrößerung des Stackbereichs) oder die fehlende Seite von der Festplatte geholt wird, falls sie vorher ausgelagert wurde.



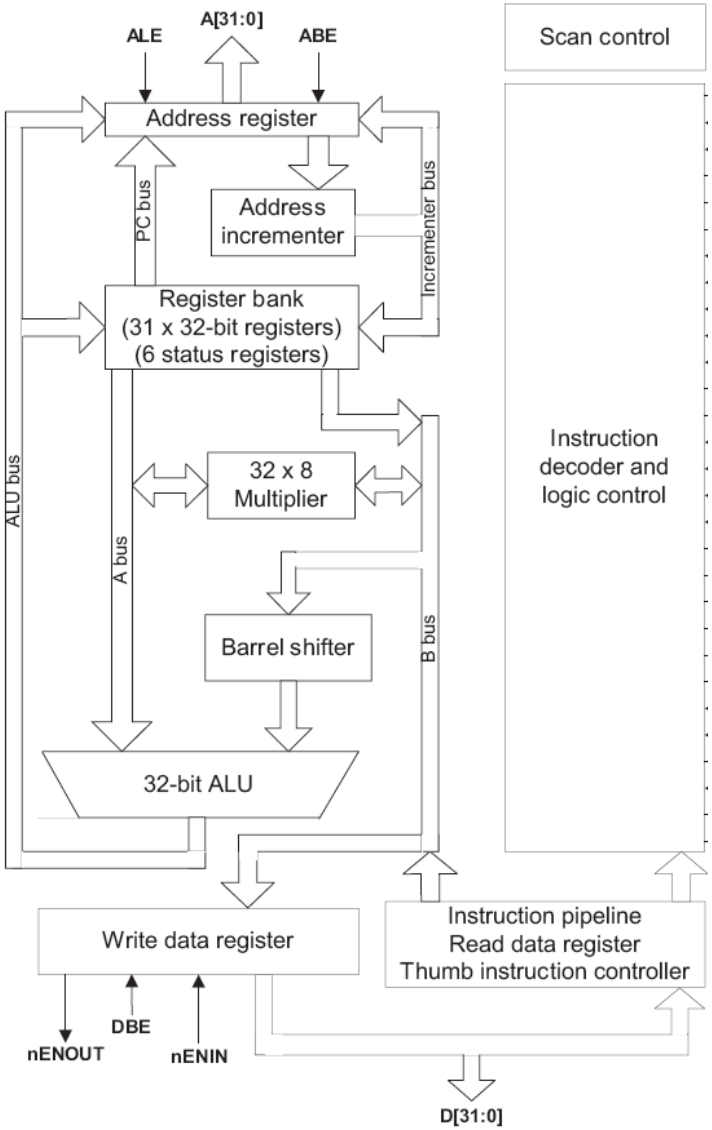
Segmentierung (Memory Protection) und Seitenverwaltung (Memory Management) sind alternative Möglichkeiten zur Speicherverwaltung und zum Speicherschutz. Segmentierung erfordert weniger Hardware- und Softwareaufwand als Seitenverwaltung.

80x86-CPUs implementieren beides, Betriebssysteme verwenden in der Regel aber nur eines der beiden Konzepte, z.B. DOS/Windows 3.1 hat mit Segmentierung gearbeitet, Windows XP/Vista und Linux benutzen Seitenverwaltung.

## 5.3 ARM-Architektur

(siehe auch http://www.arm.com)

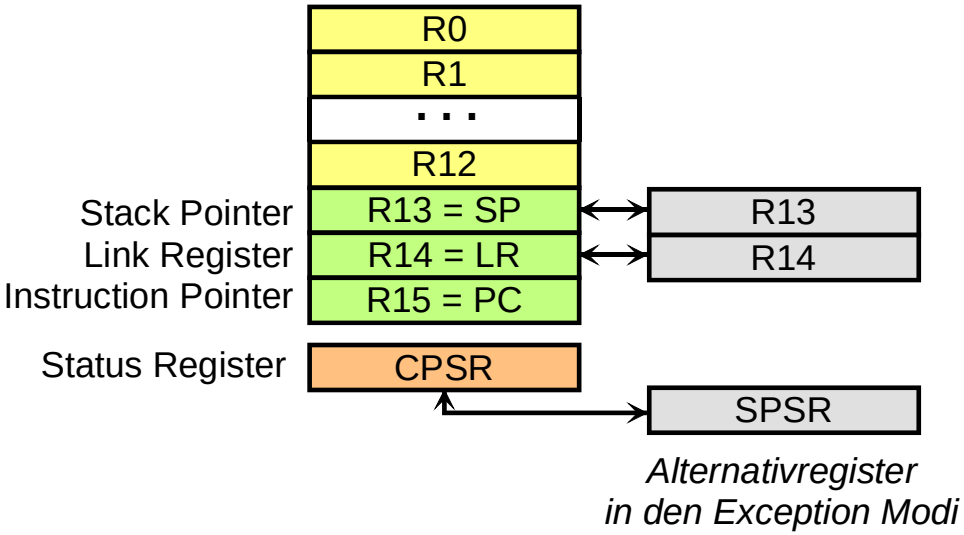
# 5.3 ARM




- Typische **32bit RISC**-Architektur
- Familie von aufwärtskompatiblen Prozessoren vom einfachen Mikrocontroller ohne Betriebssystem bis zum Mikroprozessor für Mobilcomputer
- Kleinste Ausführung **ARM 7** TDMI mit **Von-Neumann**-Speicherarchitektur, **ohne Cache, keine MPU oder MMU**, einfache **Integer-ALU**, **kurze** 3 stufige **Pipeline**
- Größere Modelle ARM 9 / 11 mit Harvard-Speicherarchitektur, Cache, MPU oder MMU in verschiedenen Ausbaustufen mit Integer- und Gleitkomma-ALUs, längere Pipeline mit 5 bis 8 Stufen, DSP- und Java-Erweiterungen
- **Advanced Risk Machine (ARM)** ist ein **Design Haus**, das die CPU-Architektur entwickelt und an Halbleiterhersteller lizenziert. Der Halbleiterhersteller ergänzt die CPU um proprietäre Peripheriekomponenten, d.h. ARM CPUs unterschiedlicher Hersteller sind nicht kompatibel, obwohl sie das gleiche Programmiermodell und dieselbe Maschinensprache verwenden.
- **Stark wachsender Marktanteil**. Gründe:

  Hohe Rechenleistung/Watt, geringe Chipgröße

## Registersatz




- **Großer Registersatz** mit 16 Registern, (3 als Spezialregister (PC, LR, SP) genutzt) und ein Statusregister (CPSR)
- Alle Register (auch PC und SP, aber nicht CPSR) können als Operanden in beliebigen Befehlen verwendet werden.
- Die Register SP und LR werden beim Auftreten eines Interrupts (Exception) umgeschaltet und eine Kopie des Statusregisters angelegt. Nach Ende des Interrupts wird auf die ursprünglichen Register zurückgeschaltet.

## Typische RISC Load- und Store-Architektur mit 3 Adress-Befehlen

- Nur ca. 40 Befehle, alle ARM Befehle 32bit lang → *einfacher Befehlsdekoder*
- Arithmetisch-logische Operationen sind nur mit Registern oder Konstanten möglich.
- In jedem arithmetisch-logischen Befehl sind 2 Operanden- und 1 Ergebnisregister möglich

  Bsp.: **`ADD R0, R1, R2`** **`R0 = R1 + R2`** *Statusbits werden nicht gesetzt*
- Bei jedem Befehl lässt sich festlegen, ob die Statusbits (Negative, Zero, Carry, ...) im Statusregister gesetzt werden sollen oder nicht.

  Bsp.: **`ADDS R0, R1, R2`** **`R0 = R1 + R2`** *Statusbits werden gesetzt*

- Bei jedem Befehl lässt sich eine Bedingung angeben. Der Befehl wird dann nur ausgeführt, wenn die Bedingung erfüllt ist, andernfalls stattdessen ein NOP ausgeführt → *vermeidet in vielen Fällen Sprungbefehle und damit Neuladen der Pipeline*

  Bsp.: **ADDMI R0, R1, #4** **R0 = R1 + 4** *wird nur ausgeführt, wenn das Negative Bit (MI=Minus) vor der Operation gesetzt war*

  **ADDMIS R0, R1, #4** *wie oben, setzt aber Statusregister*

  Mögliche Bedingungen:

  **EQ/NE** Equal/Not Equal, **CS/CC** Carry Set/Clear, **MI/PL** Minus/Positive or zero, **VS/VC** Overflow Set/Clear, **GE/GT/LE/LT** Greater Equal/Greater/Less Equal/Less (für 2er-Komplement-Zahlen), **HI/LS** Higher/Lower or same (für Betragszahlen)

- Der zweite Operand kann optional vor der Operation nach links oder rechts verschoben werden, d.h. mit einer 2er-Potenz multipliziert oder dividiert werden. Die Anzahl der Schiebeschritte kann als Konstante oder durch ein weiteres Register vorgegeben werden.

  Bsp.: **ADD R0, R1, R2 ASL #4** **R0 = R1 + (R2<<4) = R1 + R2 · $2^4$**

  **ADD R0, R1, R2 ASR R3** **R0 = R1 + (R2>>R3)= R1 + R2 / $2^{R3}$**

  mit **ASL/ASR** Arithmetisch links/rechts schieben (für 2er-Komplement-Zahlen), **LSL/LSR** Logisch links/rechts schieben (für Betragszahlen), ROR Rotieren nach rechts

- Beim Kopieren von einem Register in ein anderes lassen sich ebenfalls Bedingung, Setzen des Statusregisters und eine Schiebeoperation angeben

  Bsp.: **MOVEQS R0, R1 ASL #4** **R0 = R1<<4**, falls Z Bit gesetzt war, setzt Statusregister

## Adressierungsarten für das Laden und Sichern von Registern von/zum Speicher

- **LDR Rx, <QuellAdresse>** Laden eines Registers **Rx, x=0 … 15**
- **STR Rx, <ZielAdresse>** Sichern eines Registers **Rx, x=0 … 15**

Auch bei den Load- und Store-Befehlen kann eine Bedingung angegeben werden.

Die Quell- oder Zieladresse **<…Adresse>** im Speicher kann folgendermaßen angegeben werden: *Direkte Adresse*

*Register-indirekte Adresse*, z.B.

| | |
|---|---|
| **[R0]** | Adresse in R0 |
| **[R0,#4]** | Adresse R0 + 4 |
| **[R0,R1]** | Adresse R0 + R1 |
| **[R0,R1,ASL #2]** | Adresse R0 + (R1<<2) |

*Pre-Indexed*:

In allen obigen Beispielen wird der Inhalt von R0 und R1 nicht geändert. Setzt man dagegen ein **!** (Register Writeback) hinter die Klammer, z.B. **[R0, R1]!**, so wird **R0** mit dem berechneten Adresswert, im Beispiel also **R0=R0+R1,** überschrieben.

*Post-Indexed*:

Schreibt man die **[ ]** (ohne !) nur um das erste Adressregister, z.B. **[R0],R1,ASL #2**, so wird nur der Inhalt des ersten Adressregisters als Adresse verwendet, im Beispiel also **R0**, das Adressregister wird aber anschließend wie vorher überschrieben, im Beispiel **R0=R0+(R1<<2)**.

- Mit **LDM** und **STM** ist das gleichzeitige Laden und Sichern mehrerer Register möglich.

## Unterprogrammsprünge, Link-Register LR und Stack

- ARM-Prozessoren arbeiten standardmäßig ohne Stack. Bei einem Unterprogrammaufruf mit **`BL subroutine`** (**BL** = Branch and Link) wird die Rücksprungadresse im Register **R14=LR** gespeichert. Der Rücksprung aus dem Unterprogramm erfolgt (statt eines nicht vorhandenen RETURN-Befehls) mit **`MOV PC,LR`**, d.h. Kopieren der Rücksprungadresse aus **BL** in den Instruction Pointer **PC**.
- Ruft man innerhalb des Unterprogramms ein weiteres Unterprogramm auf, wird **R14=LR** durch die neue Rücksprungadresse überschrieben, so dass eine Rückkehr zum ursprünglichen Programm nicht mehr möglich ist. Daher muss die ursprüngliche Rücksprungadresse vorher manuell gesichert und anschließend wieder restauriert werden.
- Mit Hilfe der Befehle
  **`STR R0,[SP, #-4]!`** Dekrementiert **SP** und legt **R0** auf Stack
  **`LDR R0,[SP],#4`** Holt **R0** vom Stack und inkrementiert **SP**

  kann man die PUSH/POP/PULL-Stackoperationen üblicher Mikroprozessoren nachbilden.

  Alternativ kann man mit
  ```
  STMFD SP!, {R0-R4, R6}
  LDMFD SP!, {R0-R4, R6}
  ```
  durch einen einzigen Befehl eine in {…} angegebene Liste von Registern auf den Stack sichern bzw. restaurieren (bei Bedarf auch **LR** bzw. **PC**).
- Einfache Programmsprünge erfolgen mit **`B adresse`** (oder alternativ mit **`MOV PC,…`**).
- Alle Sprungbefehle können wieder mit Bedingungen versehen werden (Bedingte Sprünge).

## Schutzmechanismen

- Wie 80x86-Prozessoren kennen ARM-Prozessoren privilegierte und nicht privilegierte Betriebsarten, bei ARM *User Mode*, *System Mode* sowie diverse *Exception Modes*.
- Das Umschalten der Modi erfolgt durch Setzen von Bits im Statusregister bzw. Automatisch beim Auftreten eines Interrupts. Dabei wird gleichzeitig auch das Statusregister CSPR gesichert und die Register LR und SP umgeschaltet, so dass jeder dieser Modi mut seinem eigenen Stack arbeitet. Bestimmte Befehle, z.B. die Modus-Umschaltung, sind nur in den privilegierten Modi zulässig.
- Speicherschutz (Memory Protection MPU) und virtuelle Speicherverwaltung (Memory Management MMU) sind optional und nur in den größeren CPU-Varianten vorhanden.

## Vor- und Nachteile des 32bit RISC-Befehlssatzes gegenüber 16bit Mikrocontrollern

- Die durchgängige Verwendung von 32bit breiten Befehlen (*ARM-Befehlssatz*) führt zu schnellen, aber erheblich längeren Programmen als bei typischen 8/16bit Mikrocontrollern.
- Um den Speicherbedarf zu reduzieren, haben neuere ARM-Prozessoren zusätzlich einen Satz von 16bit breiten Befehlen (*Thumb-Befehlssatz*). Thumb-Programmcode ist ca. 30% kürzer als ARM-Programmcode, läuft aber ca. 40% langsamer.
- Die Umschaltung zwischen ARM und Thumb-Befehlen kann nur beim Aufruf bzw. Rücksprung von Unterprogrammen erfolgen (*ARM-Thumb-Interworking*)
- Die kürzeren Thumb-Befehle sind durch eine Reihe von Einschränkungen möglich:
  - Nur zwei statt drei Register-Operanden je Befehl (2-Adress-Befehle)
  - Keine bedingte Ausführung von Befehlen