# ARM9 Kern

# Architektur des Prozessor-Kerns #1

## Wie tiefe Kenntnisse muss der netX-C-Programmier haben?

- Viele Design-Merkmale bereits durch den Compiler berücksichtigt
- Grundverständnis notwendig, um Programme korrekt zu designen

# Architektur des Prozessor-Kerns #2

## ARM-Kern-Kodierung: ARM$d_1 d_{23} l_1 l_2 l_3$

- $d_1$ CPU-Typ
  ARM926EJ-S (200MHz)
- $d_{23}$ Zusätzlicher Speicher, weitere Funktionen
  ARM926EJ-S:
  - memory management unit
  - on-chip-Cache für Daten und Befehle
  - write buffer
  - tightly coupled memory (TCM) … schnelle SRAM nahe CPU
- $l_1$ ARM926EJ-S enhanced instruction set (für DSP-Funktionen)
- $l_2$ ARM926EJ-S Hardware-Support für Java
- $l_3$ ARM926EJ-S synthesisable – Hardware Design auf andere Si-Technologien übertragbar

## Geeignet

- General purpose
- DSP
- real-time

# Architektur des Prozessor-Kerns #3

**RISC**

- im Vgl. mit CISC muss mehr Befehle abgearbeitet werden, um gleiche Arbeit zu machen =>
  Taktrate ist bei RISC Feature Nr. 1

**Harvard** statt **von Neumann**

- < ARM 7 … von Neumann
- > ARM 9 … Harvard:
  Busse für Instruktionen und für Daten **getrennt**

Folge: ARM 9 um 30% **schneller** als ARM7 bei gleichem CLK

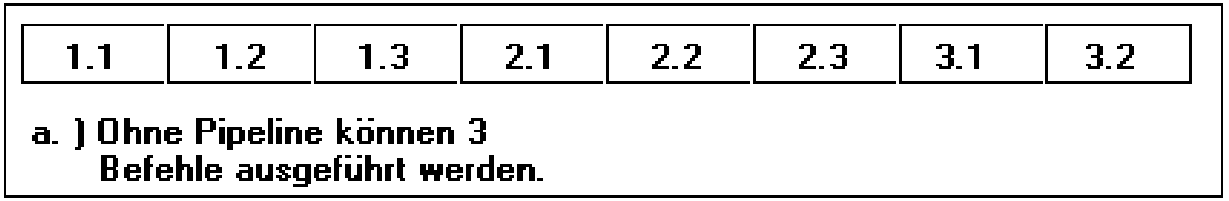
# Pipeline

3 Phasen der Instruktionsbearbeitung:

1. Fetch
2. Decode
3. Execute

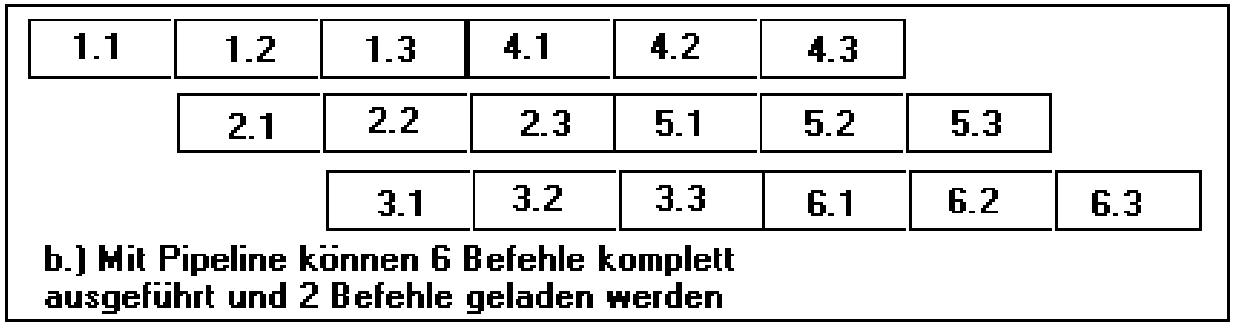
MCU ohne Pipeline:

nur nacheinander




3 Stage Pipeline:

3fach parallel




**CISC:** Befehle dauern unterschiedliche Anzahl von Takten – Pipeline geht, aber schwierig

**RISC:** 80% Befehle haben nur 1 Takt – geeignet für Pipeline

# 3-Stage-Pipeline

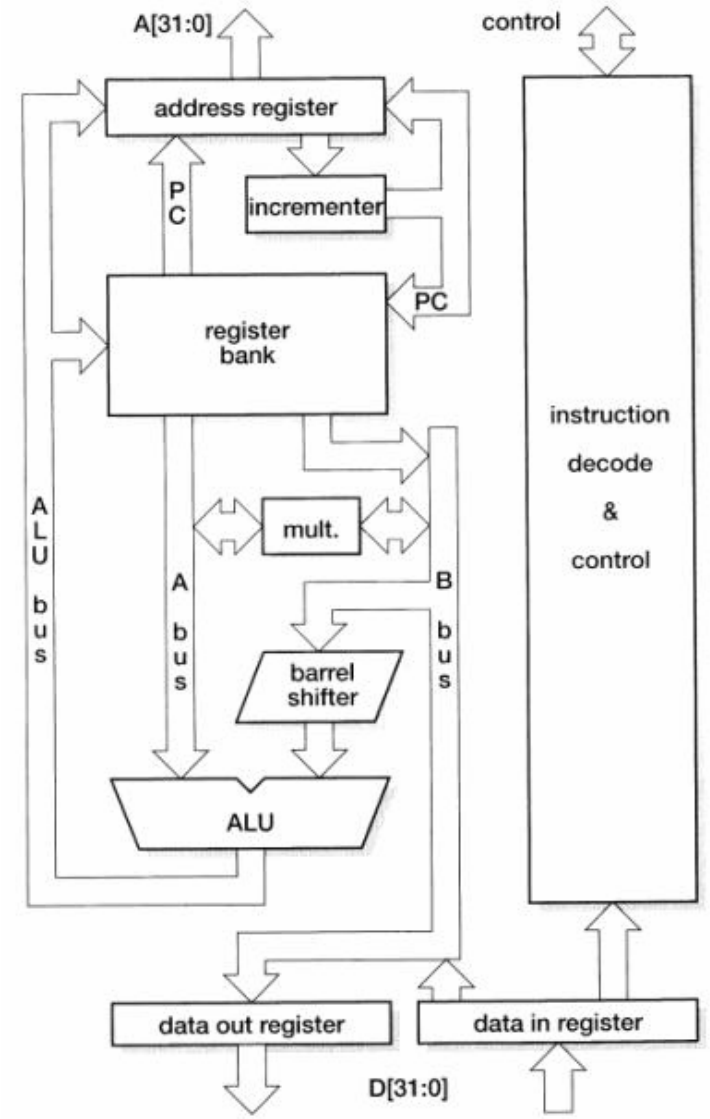



**Figure 4.1** 3-stage pipeline ARM organization.

- register bank

  speichert den Prozessor Status.
  Enthält: - 2 Leseports, 1 Schreibport
  - 1 Lese und Schreib Port, zum Zugriff auf den PC (=R15)

- barrel shifter

  hier kann ein Operand um eine beliebige Anzahl von Bits rotiert oder verschoben w.

- ALU

  Durchführung der arith. und log. Op.

- address register and incrementer

  Auswahl und Speicherung aller Adressen

- data registers (in/out)

  Speicherung aller Daten, die von und zu dem Speicher transportiert werden

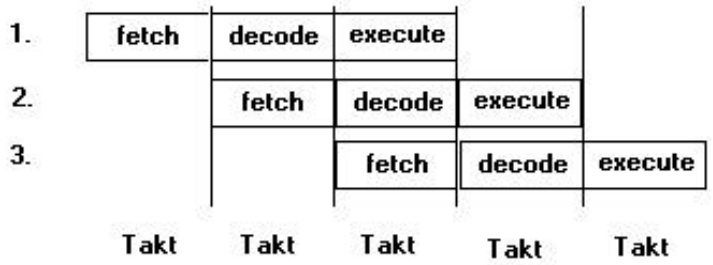
- instruction decoder and associated control logic

# 3-Stage-Pipeline

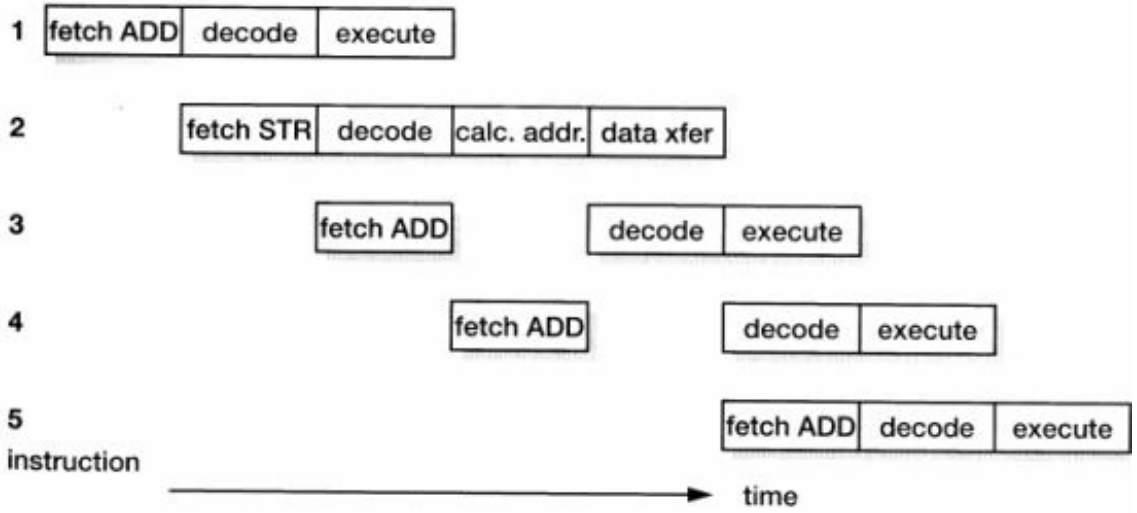
3 Stage Pipeline (<=ARM7) – auch 1 Takt Befehle in 3 Phasen unterteilt:

- **Fetch** Befehl wird aus dem Speicher geladen und in Pipeline eingefügt
- **Decode** Befehl wird decodiert
- **Execute** Befehl wird ausgeführt (Regs lesen, Daten in ALU kombinieren, Daten in Zielregs zurück

Ablauf bei single-cycle Befehlen




Ablauf bei multi-cycle Befehlen




Folge: der Programm-Zähler läuft dem gerade ausführenden Befehl vor

# 5-Stage-Pipeline

- **Fetch**

Befehl wird aus dem Speicher geladen und in Pipeline eingefügt

- **Decode**

Befehl wird decodiert, OPs aus Regs lesen. Zugriff in 1 Takt dank Operand ReadPorts

- **Execute**

Die ALU berechnet das Ergebnis. Falls **load** oder **store,** berechnet die ALU die Speicher Adresse

- **Buffer/Data**

Operationen auf dem Datenspeicher finden statt. Wenn keine **benötigt werden** wird das Resultat der ALU einen Takt gepuffert

- **Write-Back**

Die Resultate des Befehls werden in die Register zurück geschrieben, durch alle Daten die vom Speicher gelesen worden sind.

# 5-Stage-Pipeline

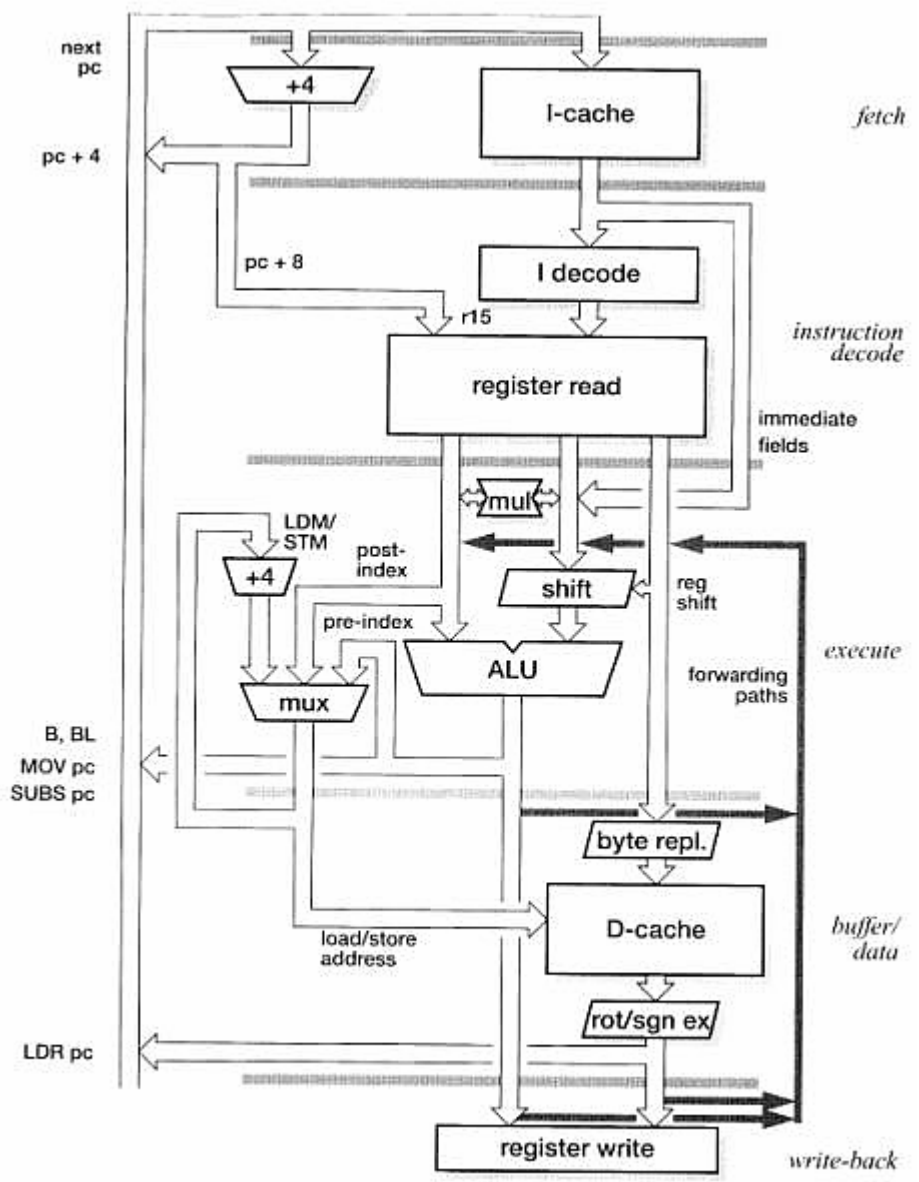


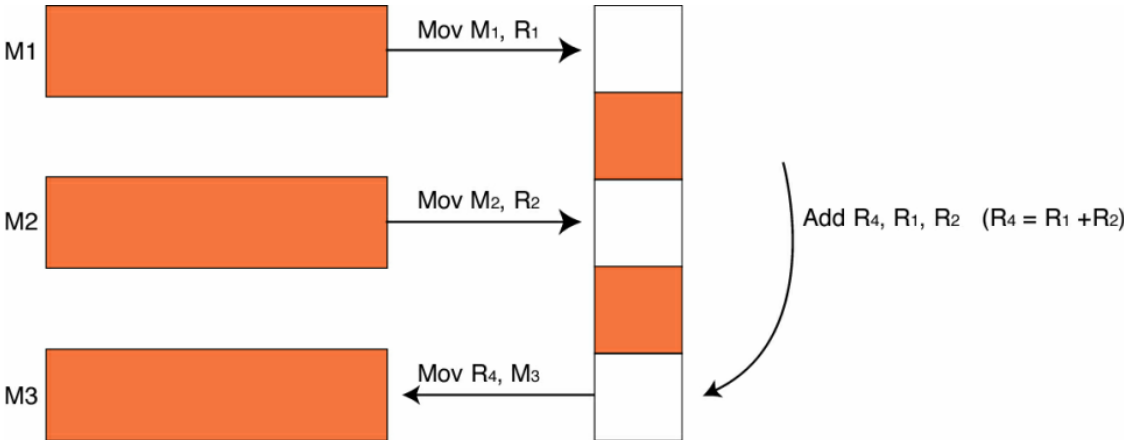

**Figure 4.4** ARM9TDMI 5-stage pipeline organization.

**Klassich:**
Regblock: 2 Lese-, 1 Schreibport

**Besonderheiten ARM:**
- Regblock: 3 Lese-, 2 Schreibports
- Execute Stage: mehrfache Speicherzugriffe möglich
- „Data Forwarding“:
  Falls das Ergebnis der vorherigen Befehls zur Ausführung des nächsten notwendig -> aus jeder Stage möglich, die Ergenbisse zu lesen
  Siehe: Immediate fields

# CPU Registers

ARM9: load-and-store Architektur

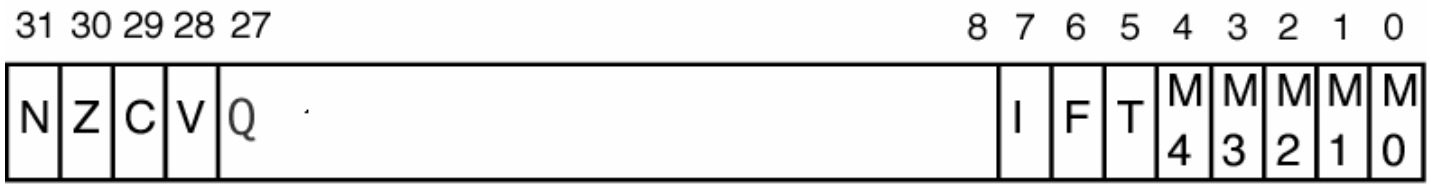



1. Speicher -> CPU regs
2. Befehl Ausführung
3. Erg. CPU reg->Speicher

# CPU Registers: Register Stack


- 16 32-Bit Register
- R0-R12 … general purpose
- Spezielle Funktionen:
  Stack Pointer, Link Register, Program Counter
  LR: speichert die Rücksprungadresse von Funktionen
  weitere Funktionsaufrufe: Rücksprungadr. auf dem Stack
- CPSR – Current Program Status Register

# CPSR-Current Program Status Reg




- negative, zero, carry, overflow
- Q = saturation flag, verwendet bei DSP. Sticky, zu löschen nur per Appl. Code
- Bits 7-0 manipulierbar per Application Code:

  I, F … Erlaubnis externer Interrupts (disable = 1 !)

  T … Thumb bit:

  ARM: „normally“ instruction set (32 Bit) / thumb instruction set (16 Bit)

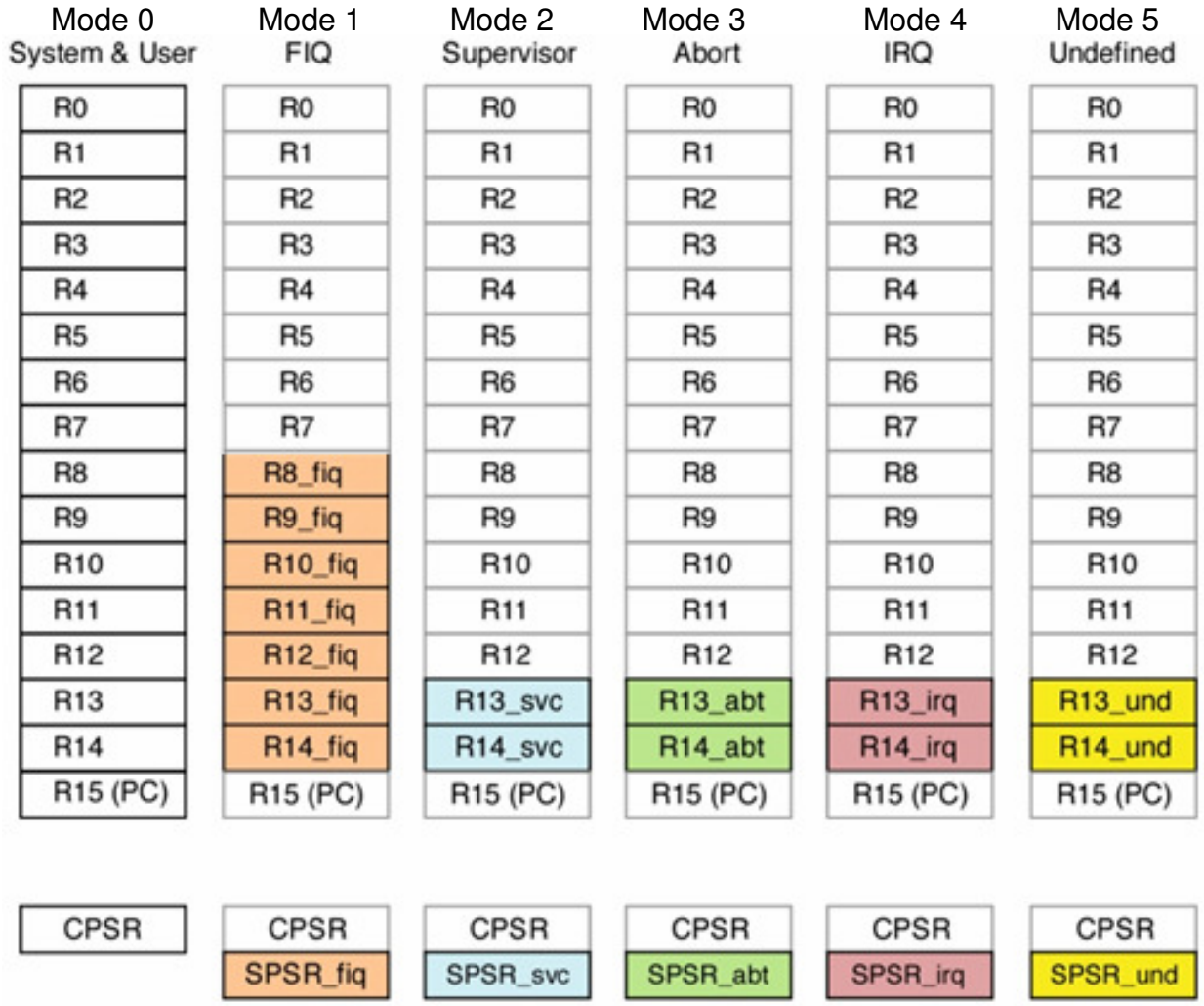
M4-M0 … Mode bits – definieren, in welchem der 6 operation modes sich die CPU befindet.

## Operation modes:

normally operation -> User Mode

Antwort auf Exception (int, mem err, sw int) -> Übergang in ein Spezialmodus

# Operation Modes




**Modi 2-5:**
SP, LR dupliziert

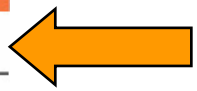
**Mode 1:**
R8-R14 dupliziert
(diese Regs müssen bei FIQ nicht gerettet werden)

**Modi 1-5:**
Kopie von CPRS vom UserMode gerettet in SPSR (saved PSR)

# Exception Modes

In welchen Operation Mode übergeht die CPU bei welcher Exception?

| Exception | Mode | Address |
|---|---|---|
| Reset | Supervisor | 0x00000000 |
| Undefined instruction | Undefined | 0x00000004 |
| Software interrupt (SWI) | Supervisor | 0x00000008 |
| Prefetch Abort (instruction fetch memory abort) | Abort | 0x0000000C |
| Data Abort (data access memory abort) | Abort | 0x00000010 |
| IRQ (interrupt) | IRQ | 0x00000018 |
| FIQ (fast interrupt) | FIQ | 0x0000001C |



Ab diesen Adressen läuft es weiter

(0x14 fehlt wg. Kompatibilität mit Vorgängerversionen)

| Priority | Exception |
|---|---|
| Highest 1 | Reset |
| 2 | Data Abort |
| 3 | FIQ |
| 4 | IRQ |
| 5 | Prefetch Abort |
| Lowest 6 | Undefined instruction SWI |

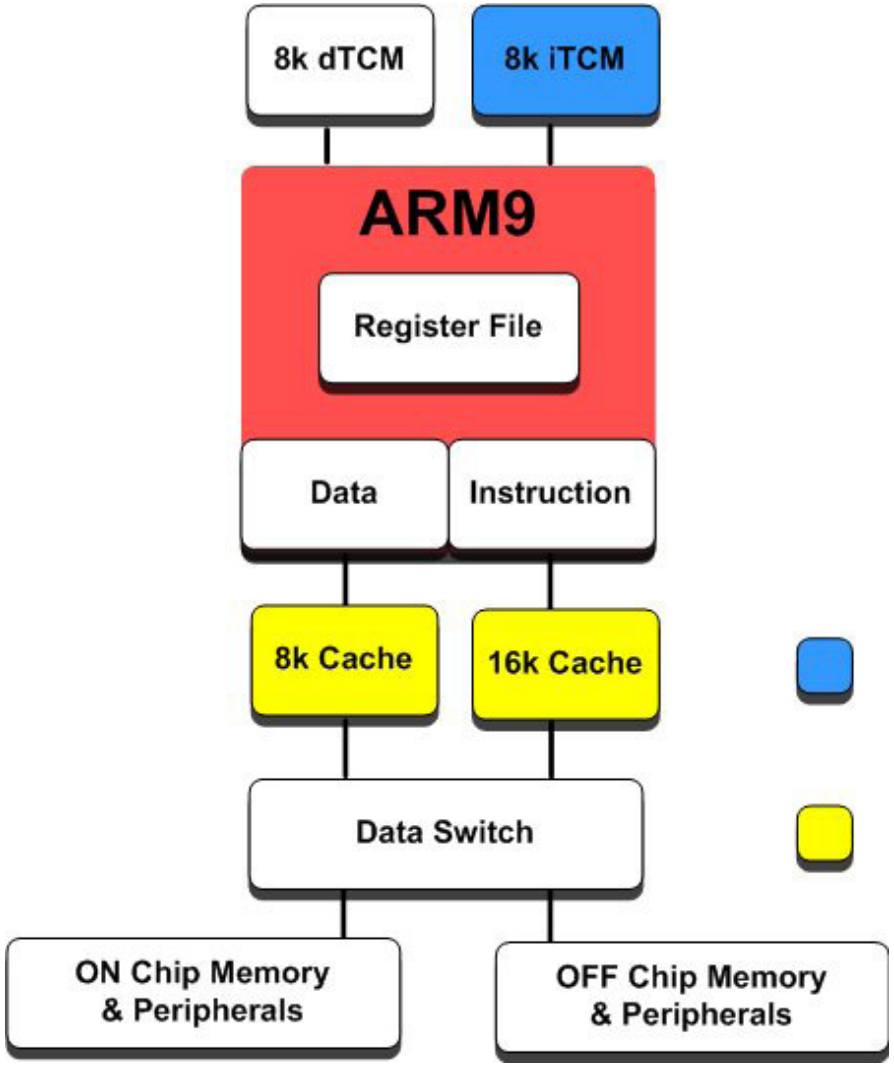
Die CPU-Int-Prioritäten sind fest

# Tätigkeit bei Exception

z.B. bei IRQ:

1. (PC+4) -> LR ((PC+4) Adr. der nächsten Instr.)
2. CPSR -> SPSR_irq
3. 0x18 -> PC
   User_Mode -> IRQ_Mode
4. CPSR: I=1 (IRQ disabled)
   Falls verschachtelte Ints benötigt, muss:
   1. IRQ explizit erlaubt werden
   2. LR -> (SP)
5. Innerhalb ISR: R0-R12 auf dem IRQ Stack retten
6. Am Ende von ISR:
   Bei ARM gibt es keinen „return from interrupt“ –> manuell
   Fälle:
   1. SWI:
      **LR->PC** `MOVS R15, R14`
   2. IRQ, FIQ, Prog Abort:
      **LR-4 -> PC** (wg. Pipeline) `SUBS R15, R14, #4` (inkl. Restore UserMode)
   3. Data Abort: Exception kommt 1 Befehl nach dem Befehl, der sie verursachte
      **LR-8 -> PC** (um 2 Befehle zurück) `SUBS R15, R14, #8`

# ARM9 Speicher Architektur




nur bei netX50

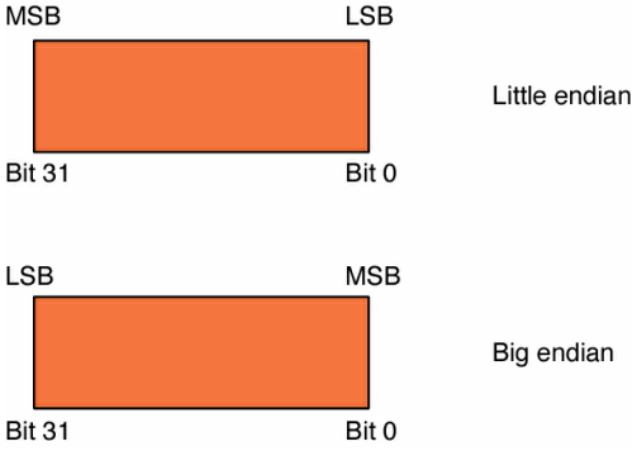
nur bei netX100 und 500

# ARM9 Instruction set

Verständnis von ASM-Ansätzen auch beim Programmieren in C wichtig

2 instruction sets:

- „normally“ 32 Bit
- thumb 16 Bit

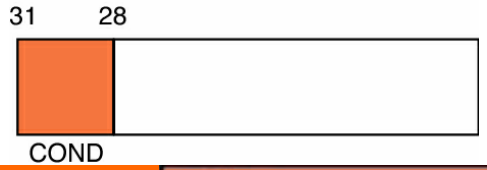



**ARM9:**
little oder big endian (im Rahmen des DWORDs)

**netX:**
nur little endian
Einstellung des Compilers prüfen!

# ARM9 Instruction set: Sprünge

Jeder ARM-Befehl kann bedingt ausgeführt werden




Die höchsten 4 Bits XORed mit CPSR condition codes
Bei Nicht-Übereinstimmung: NOP

| Prä-/suffix | Flags | Meaning |
|---|---|---|
| EQ | Z set | equal |
| NE | Z clear | not equal |
| CS | C set | unsigned higher or same |
| CC | C clear | unsigned lower |
| MI | N set | negative |
| PL | N clear | positive or zero |
| VS | V set | overflow |
| VC | V clear | no overflow |
| HI | C set and Z clear | unsigned higher |
| LS | C clear and Z set | unsigned lower or same |
| GE | N equals V | greater or equal |
| LT | N not equal to V | less than |
| GT | Z clear AND (N equals V) | greater than |
| LE | Z set OR (N not equal to V) | less than or equal |
| AL | (ignored) | always |

Bsp.
**`EQMOV R1, 0x00800000`**

0x00800000 -> R1
erfolgt nur, wenn die letzte arith. Instruktion das Z-Flag gesetzt hat

Die Präfixe sind Standard bei:

**MOV**

**ADD**

Kein Pflicht!

Geeignet für kleine Schleifen wie: **`if(x<100) x++;`**

Bei bedingten Befehlen kein Pipeline-Fush&Refill nötig => Performance

# ARM9 Instruction set: 6 Kategorien

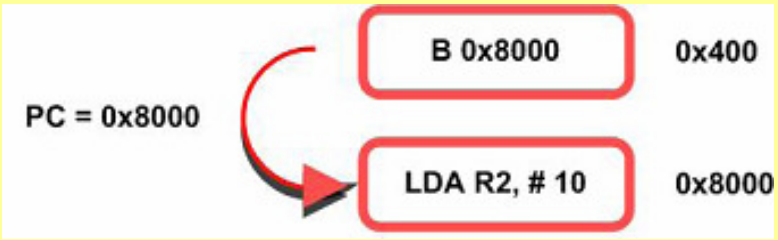
1. Branching
2. Data Processing
3. Data Transfer
4. Block Transfer
5. Multiply
6. Software Interrupt

ARM Quick Reference Card:
http://infocenter.arm.com/help/topic/com.arm.doc.qrc0001l/QRC0001_UAL.pdf

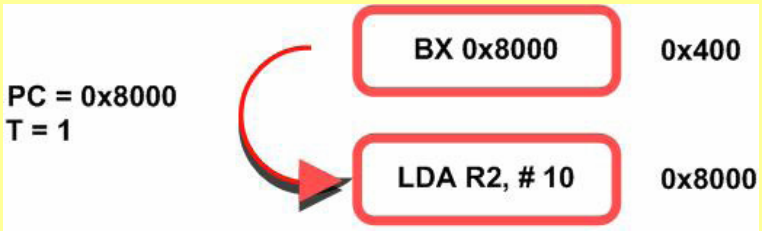
# ARM9 Instructions: 1.Branching

Ohne Suffix: Unbedingte Sprünge
Mit Suffix: Bedingte Sprünge

**Branch** (B <label>, BEQ <label>,…)




**Branch exchange**




Ohne Suffix: Unbedingte Funktionsaufrufe
Mit Suffix: Bedingte Funktionsaufrufe

**Branch link** (R14=Link Register)

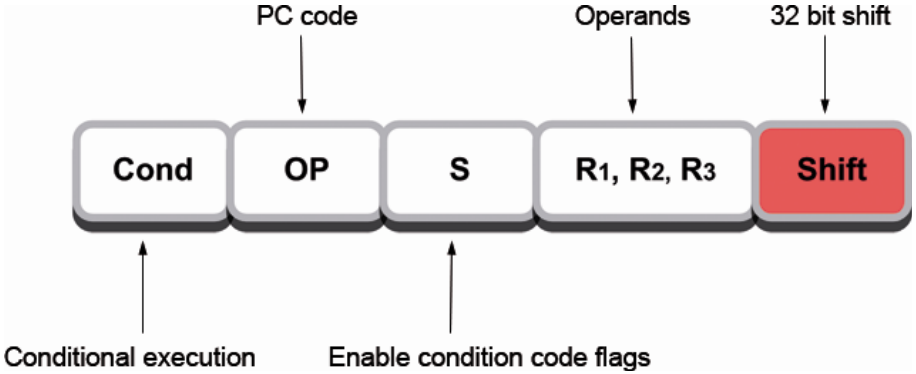

**Branch exchange link**


Wie oben, zusätzlich Umschaltung der Befehlsätze ARM <-> THUMB

label = ±32 MB

# ARM9 Instructions: 2.Data Processing

Befehlsmnemonik




Shift: mittels Barrel Shifter

| | |
|---|---|
| Cond | Identitätsprüfung mit Condition Flags (NZCV) in CPRS |
| OP | Befehlskürzel |
| S | log.1=Ergebnis des Befehls beeinflusst Condition Flags |
| R1 | Ergebnis |
| R2 | Operand1 (nur Register) |
| R3 | Operand2 (Register oder Immediate) |
| Shift | Operand 2 kann um max.32 bit (gegeben durch immediate Wert) versch |

Bsp:

```
EQADDS R1,R2,R3,LSL #2          ; if (Z==1) R1=R2+(R3*4)
```

# ARM9 Instructions: 2.Data Processing

## Beispiele

| | |
|---|---|
| AND | Logical bitwise AND |
| EOR | Logical bitwise exclusive OR |
| SUB | Subtract |
| RSB | Reverse Subtract |
| ADD | Add |
| ADC | Add with carry |
| SBC | Subtract with carry |
| RSC | Reverse Subtract with carry |
| TST | Test |
| TEQ | Test Equivalence |
| CMP | Compare |
| CMN | Compare negated |
| ORR | Logical bitwise OR |

| | |
|---|---|
| MOV | Move |
| BIC | Bit clear |
| MVN | Move negated |

# ARM9 Instructions: 3.Data Transfer #1

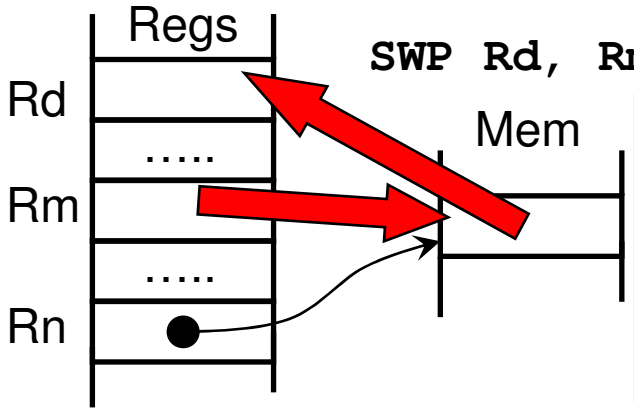
## Transfer allgemein - Beispiele

| | | | |
|---|---|---|---|
| LDR | Load Word | STR | Store Word |
| LDRH | Load Half Word | STRH | Store Half Word |
| LDRSH | Load Signed Half Word | STRSH | Store Signed Half Word |
| LDRB | Load Byte | STRB | Store Byte |
| LRDSB | Load Signed Byte | STRSB | Store Signed Half Word |

Bsp:

```
LDR R15,[R2], #4                ; PC = [R2] + 4
```

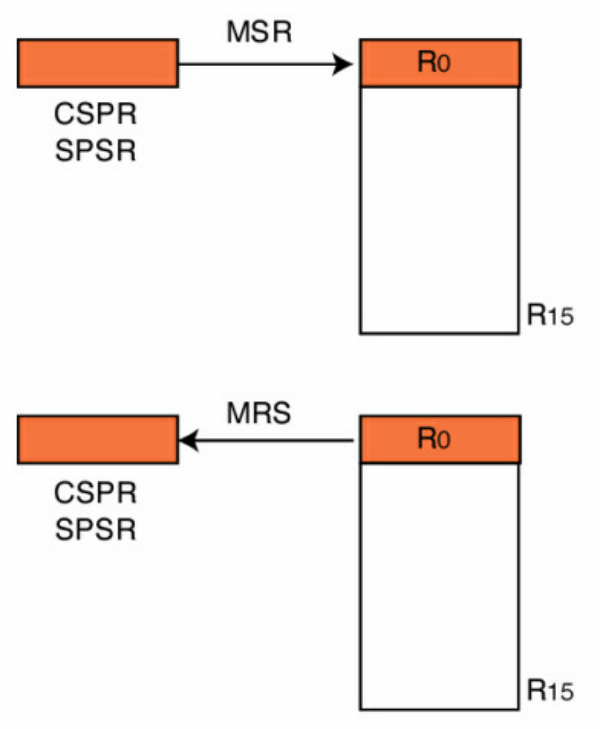
**Speziell: Swap** (nicht unterstützt durch C)




1 Befehl:
Kann durch Interrupts nicht Unterbrochen w.
=> Real-time

# ARM9 Instructions: 3.Data Transfer #2

## Speziell: Status CPSR und SPSR ändern

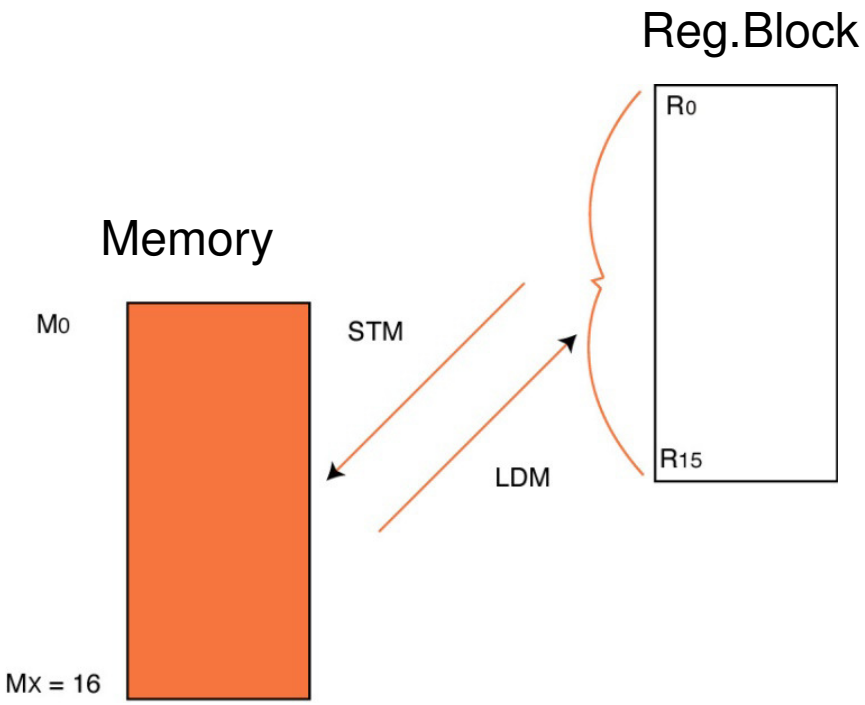



Bsp. IRQ disablen
(einzelne Bits können nur in Regs geändert w.)

MSR, MRS funktionieren nicht im User Mode =>

1. In den gewünschten Mode switchen – dies geht nur durch Exception, Reset, IRQ, FIQ, SWI
2. Änderung via MSR/MRS vornehmen
3. In den User Mode zurück

# ARM9 Instructions: 4.Block Transfer




Speichern/Laden

- alle Register
- Registerblock

mit einem einzigen Befehl

# ARM9 Instructions: 5. Multiply

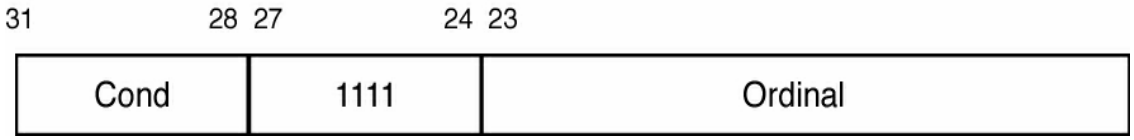
MAC=Multiply Accumulate Unit
ARM9: integer and long integer multiplication

## Beispiele

MULA  32x32+32 = 32 MAC
UMULL 32x32 = 64 Unsigned multip
UMLAL 32x32+64 = 64 Unsigned MAC
SMULL 32x32 = 64 Signed multiply
SMLAL 32x32+64 = 64 Signed MAC

# ARM9 Instructions: 6.SW-Interrupts

1. Exception wird generiert
2. CPU in Supervisor mode
3. PC ← 0x8




```
SWI #3          ; Wert 3 wird in den unbenutzten Teil – Bits 0-23 kopiert
```

Die ordinal Bits können in SWI ISR dekodiert werden; Benutzung:

- calls  in the protected mode
- privileged code ausführeren
- **BS Aufrufe tätigen**

24 ordinal Bits => 224 = 16 777 216 SWI ISR

# Thumb instruction set

THUMB = komprimierte 16-Bit Befehle

- Üblich: Mix aus ARM und THUMB Instruktionen => interworking process
- Platzeinsparung im Programm-Speicher: 30%
- Verlangsamung um 40%
- Block der verfügbarer Register reduziert auf 8 (R0-R7)
- THUMB-Befehle erinnern an klassische MCU-Befehle (z.B. nur 2 Regs):

| ARM | THUMB | Bedeutung |
|---|---|---|
| `ADD R0, R0,R1` | `ADD R0,R1` | `R0 = R0+R1` |

Nur **`MOV, ADD, CMP`** arbeiten mit R8-R12 (restricted access register)

- CPSR/SPSR nur indirekt manipulierbar
- BX, BLX – Umschalten ARM <->THUMB Modus
  (Nach Reset in Exception Mode nur ARM)
- Nur in THUMB: PUSH, POP (Stack bottom = [R13] )
- SWI: Ordinal Bits begrenzt auf 8 (max. 256 SWI ISR)