6.10 Erweiterte Speicherzugriffe

6.10.1 Hintergrund

In den meisten Programmen werden folgende Operationen durchgeführt

- Bearbeitung von Strings
- Zugriff auf Felder (einfache/komplexe Datentypen)
- Unterprogrammsprünge

Für die effiziente Durchführung dieser Operationen besitzt der ARM erweiterte Adressierungsarten.

6.10.2 Speicher laden mit konstantem Offset zu einer Basisadresse

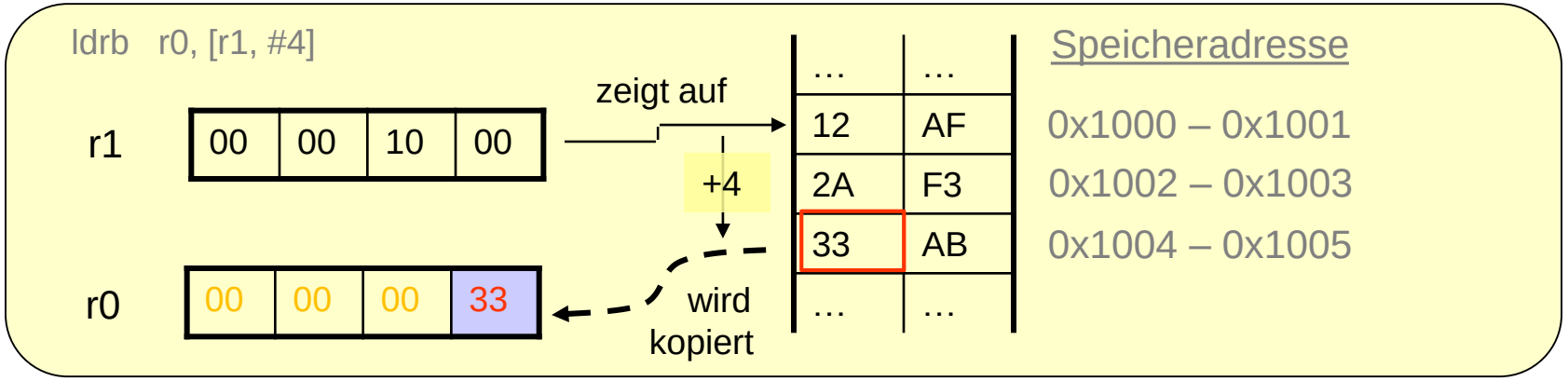
Aufruf: ldr Zielregister, [Basisadressregister, #Offset]

Quelladressierung : *preindex (immediate offset)*

Arbeitsweise und Besonderheiten:

- [Zielregister] ← [M([Basisadressregister] + Offset)]
- **ldr** lädt Wort, **ldrb** lädt Byte (die vorderen 3 Byte des Registers werden mit 0 besetzt)
- Der Offset muss im Bereich -255 … +4095 liegen.
- Nützlich bei Zugriff auf Tabellen, Unterprogrammparameter, Datenfelder

- **str** speichert Wort, **strb** speichert ein Byte



6.10.3 Speicher laden mit variablem Offset zu einer Basisadresse

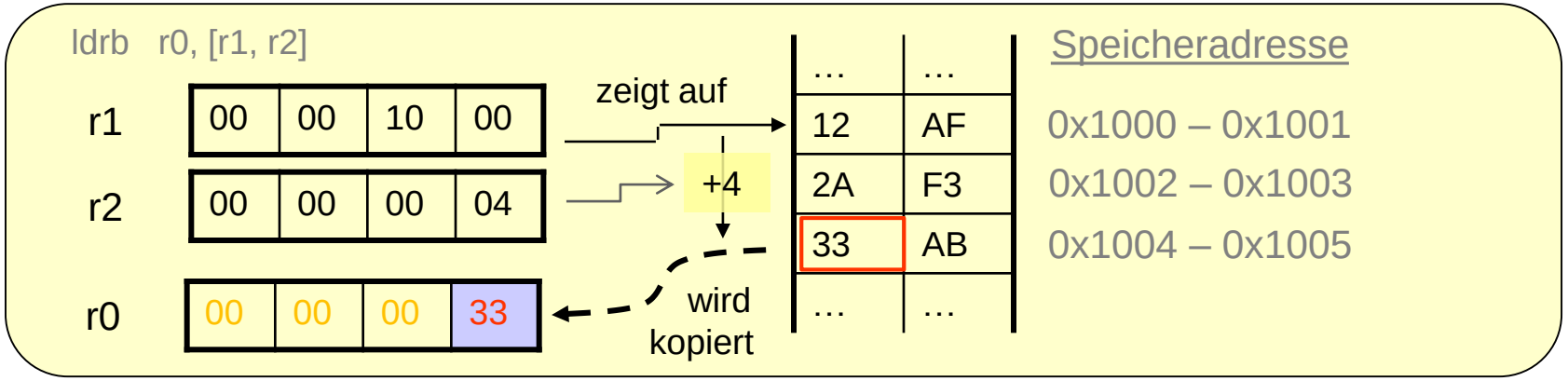
Aufruf: ldr Zielregister, [Basisadressregister, Offsetregister]

Quelladressierung : *preindex (register offset)*

Arbeitsweise und Besonderheiten:

- [Zielregister] ← [M([Basisadressregister] + [Offsetregister])]
- **ldr** lädt Wort, **ldrb** lädt Byte (die vorderen 3 Byte des Registers werden mit 0 besetzt).
- Nützlich bei Zugriff auf Tabellen und Strings.

- **str** speichert Wort, **strb** speichert ein Byte



6.10.4 Speicher laden mit konstantem Offset, pre-indexing und update

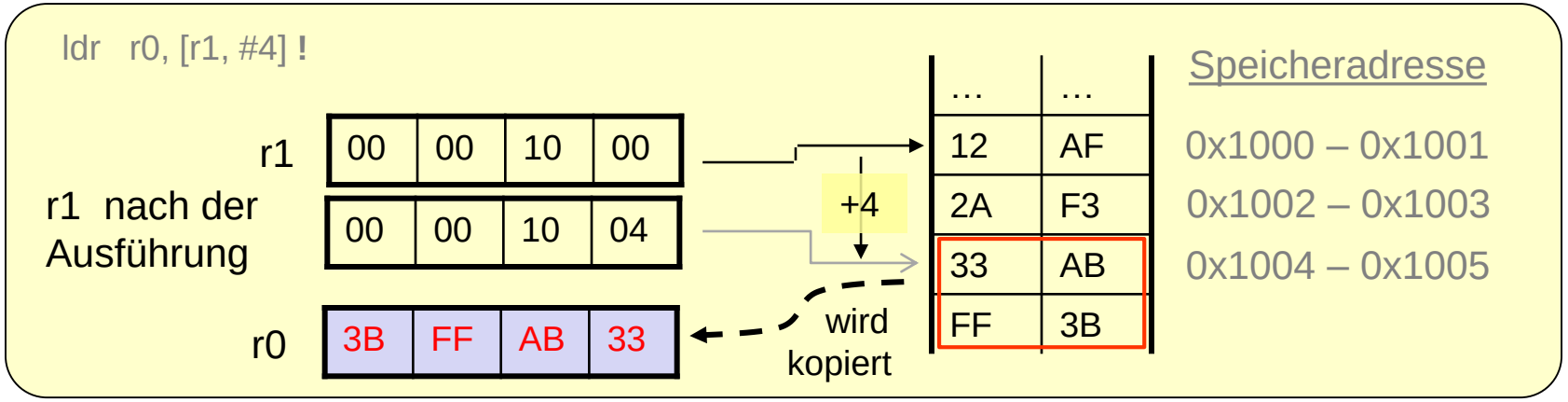
Aufruf: ldr Zielregister, [Basisadressregister, # Offset] **!**

Quelladressierung : *preindex (immediate offset) with writeback*

Arbeitsweise und Besonderheiten:

- 1. [Zielregister] ← [M([Basisadressregister] + Offset)]
  2. [Basisadressregister] ← [Basisadressregister] + Offset
- **ldr** lädt Wort, **ldrb** lädt Byte (die vorderen 3 Byte des Registers werden mit 0 besetzt)
- Der Offset muss im Bereich -255 … +255 liegen.
- Nützlich beim Abarbeiten von Tabellen/Zeichenketten und bei Stackoperationen

- **str** speichert Wort, **strb** speichert ein Byte



6.10.5 Speicher laden mit konstantem Offset, post-indexing und update

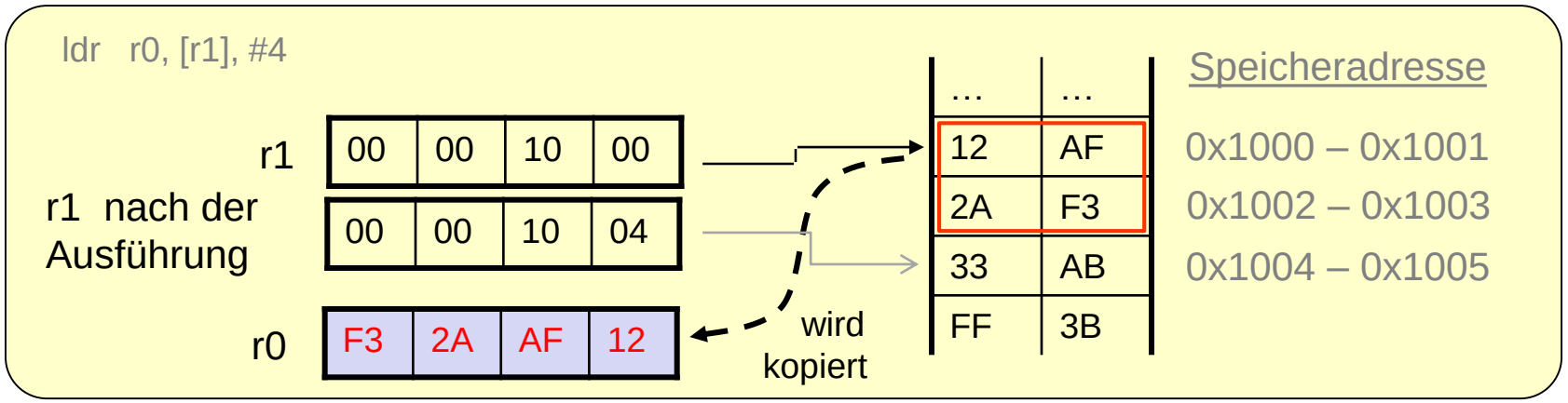
Aufruf: ldr Zielregister, [Basisadressregister] , # Offset

Quelladressierung : *postindex (immediate offset)*

Arbeitsweise und Besonderheiten:

- 1. [Zielregister] ← [M([Basisadressregister])]
  2. [Basisadressregister] ← [Basisadressregister] + Offset
- **ldr** lädt Wort, **ldrb** lädt Byte (die vorderen 3 Byte des Registers werden mit 0 besetzt)
- Der Offset muss im Bereich -255 … 255 liegen.
- Nützlich beim Abarbeiten von Tabellen/Zeichenketten und bei Stackoperationen

- **str** speichert Wort, **strb** speichert ein Byte



ÜBUNG: Adressierungsarten „ immediate offset, register offset, pre-indexed, post-indexed "

Geben Sie jeweils die Registerinhalte bzw. Speicherinhalte nach den Befehlen an.
Alle verwendeten Register sind mit 0xFFFFFFFF initialisiert.
Im Speicher stehen folgende Werte (Hex.):

Adr.	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	100A	100B
Inh.	**10**	**06**	**34**	**45**	**56**	**67**	**78**	**89**	**9A**	**AB**	**BC**	**CD**

		Reg (Bit): Mem (Byte-Adresse):	**31..24** +0	**23..16** +1	**15..8** +2	**7..0** +3
`mov`	`r0, #0x1000`	`[r0] =`				
`ldr`	`r1, [r0, #4]`	`[r1] =`				
`strb`	`r1, [r0, #2]`	`[M(0x1000)]=`				
`ldr`	`r2, [r0, #4]!`	`[r2] =`				
`ldr`	`r3, [r0, #4]!`	`[r3] =`				
`ldrb`	`r4, [r0], #1`	`[r4] =`				
`ldrb`	`r5, [r0], #1`	`[r5] =`				
`ldrb`	`r6, [r0], #1`	`[r6] =`				
`str`	`r1, [r0, #-8]`	`[M(0x1002)]=`				

6.11 Erweiterte Registeroperationen

6.11.1 Hintergrund

In vielen Programmen kommen folgende Operationen vor:

- indizierte Zugriffe auf Halbwort- und Wortfelder
- schnelle Multiplikationen mit $2^{n,}$
- Division,
- Bitmanipulationen,
- mathematische Operationen (Quadratwurzel, Logarithmus, trigonometr. Fkt.),
- Erzeugung von Zufallszahlen.

Für die effiziente Durchführung dieser Operationen besitzt der ARM einen sog. **Barrel-Shifter**.

ERGÄNZUNG / EINSCHUB

5.3 Binäre Multiplikation

5.3.1 Multiplikation mit 2

Für Binärzahlen gilt: Multiplikation mit 2 = um eine Stelle nach links schieben

verdeutlichendes Beispiel: 45*2 = 90

45

Wertigkeit	128	64	**32**	16	**8**	**4**	2	**1**
Binärzahl	0	0	**1**	0	**1**	**1**	0	**1**

45 * 2

Wertigkeit	128	**64**	32	**16**	**8**	4	**2**	1
Binärzahl	0	**1**	0	**1**	**1**	0	**1**	0

ERGÄNZUNG / EINSCHUB

[Binäre Division]

[Division durch 2-er Potenzen]

Für Binärzahlen gilt: Division durch 2 = um eine Stelle nach rechts schieben

Achtung: Inhalt der letzten Stelle(n) vor Division = Rest der Division

wieder ein Beispiel: 45 DIV 4 = 11 Rest 1

Rest = 1 → 1/4

45

Wertigkeit	128	64	**32**	16	**8**	**4**	2	**1**
Binärzahl	0	0	**1**	0	**1**	**1**	0	**1**

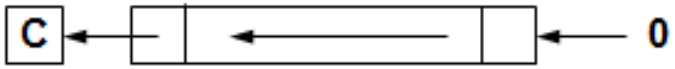
45 DIV 4

Wertigkeit	128	64	32	16	**8**	4	**2**	**1**
Binärzahl	0	0	0	0	**1**	0	**1**	**1**

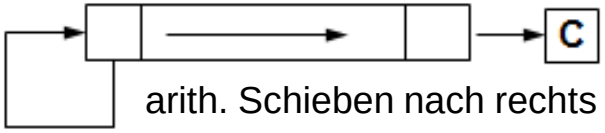
Beliebiger Divisor: Deutlich aufwändigere Rechnung

6.11.2 Barrel Shifter

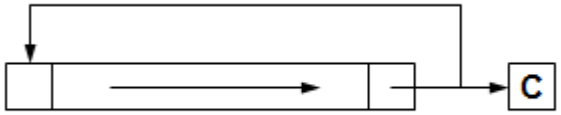
- Erlaubt Bit-Verschiebungen und –Rotationen um einen oder mehrere Bitpositionen in einem Schritt.
- Typ. Operationen sind:



log. Schieben nach links

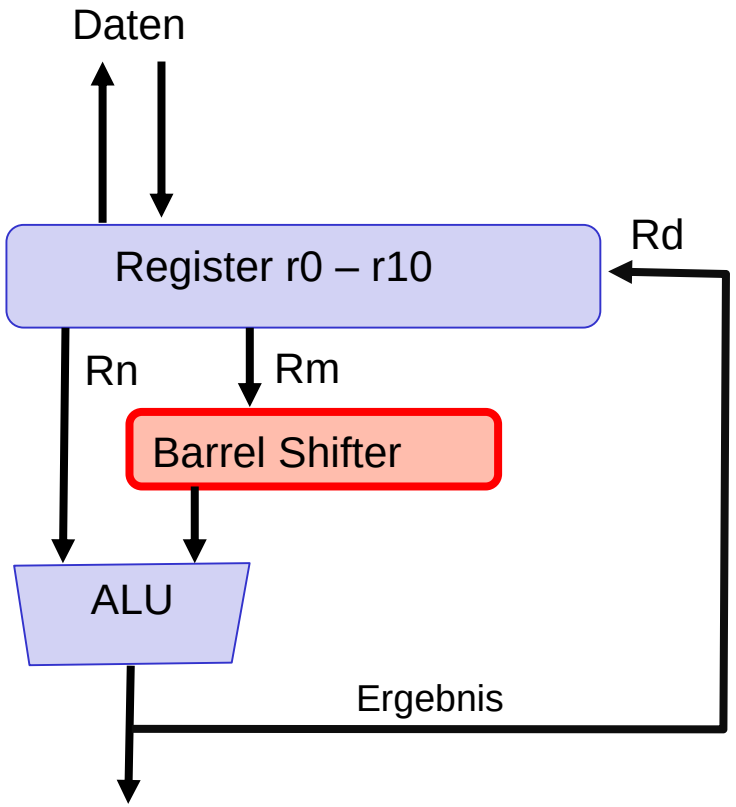


arith. Schieben nach rechts



Rotation nach rechts

Rotation nach rechts über carry



Diese Folie nur zur Info

6.11.3 Shifted-Register-Operand

Bei den meisten Registeroperationen kann der letzte Operand vor der Operation um bis zu 32 bit verschoben oder rotiert werden.

Fünf Typen von Schiebeoperationen stehen zur Verfügung:

ASR	Arithmetisches Schieben nach rechts
LSL	Logisches Schieben nach links
LSR	Logisches Schieben nach rechts
ROR	Rotieren nach rechts
RRX	Rotieren nach rechts über carry

Beispiele:

- mov r2, r0, **LSL #2** [r2] ← LSL2(r0) d.h. [r0] * 4
- add r9, r5, r5, **LSL #2** [r9] ← [r5] + LSL2(r5) d.h. [r5] * 5
- sub r0, r4, r5, **LSL #2** [r0] ← [r4] - LSL2(r5) d.h. [r4] – [r5]*4
- sub r0, r4, r5, **LSR #3** [r0] ← [r4] - LSR3(r5) d.h. [r4] – [r5]/8
- mov r2, r0, **LSL r3** [r2] ← [r0] linksverschoben um den Wert in r3

6.11.4 Befehlscodierung