

bit po 14 bits Zahl ( $-2^{13} \dots 2^{13}-1$ ) auf Position 2-15

Berechnung von  $a + \frac{b}{4}$  Ergebnis in R<sub>1</sub> (32 bits)

MOV R<sub>0</sub>, R<sub>0</sub>, LSL #16 ; bit 15 an bit 32 schieben mit carry

MOV R<sub>0</sub>, R<sub>0</sub>, ASR #18 Bit 31 zu bit 15 schieben

PSB R<sub>1</sub>, R<sub>0</sub>, R<sub>0</sub>, LSL #3 ;  $R_1 = (R_0 \leftarrow 3) - R_0 = 7R_0$  (Reverse Sub)

ADDMI R<sub>1</sub>, R<sub>1</sub>, #3 entspricht floor(a+3)/4 bei negativer Zahl

MOV R<sub>1</sub>, R<sub>1</sub>, ASR #2 dividiere 4

Addr R<sub>1</sub>, R<sub>2</sub>  $(\text{int} b_4 \dots a \dots b) \Rightarrow (a+b)/2$  Ergebnis in Addr R<sub>0</sub> (WS 17/18)

LDR R<sub>3</sub>, [R<sub>1</sub>] / LDM R<sub>1</sub>, {R<sub>3</sub>, R<sub>4</sub>}  $R_3 = a\text{-low}$   
LDR R<sub>4</sub>, [R<sub>1</sub>, #4]  $R_4 = a\text{-high}$

LDR R<sub>5</sub>, [R<sub>2</sub>]

LDR R<sub>6</sub>, [R<sub>2</sub>, #4]

ADDS R<sub>3</sub>, R<sub>3</sub>, R<sub>5</sub>  $R_3 = a\text{-low} + b\text{-low}. \text{ Set carry}$

ADC R<sub>4</sub>, R<sub>4</sub>, R<sub>6</sub>  $R_4 = a\text{-high} + b\text{-high} + \text{Carry}$

MOV R<sub>5</sub>, R<sub>4</sub>, LSR #1  $R_5 = R_4 \gg 1$  (high)

MOV R<sub>6</sub>, R<sub>3</sub>, LSR #1  $R_6 = R_3 \gg 1$  (low)

ORR R<sub>6</sub>, R<sub>6</sub>, R<sub>4</sub>, LSL #31 schreiben bit 0 von R<sub>4</sub> an bit 31 von R<sub>6</sub>

STR R<sub>6</sub>, [R<sub>0</sub>] speicher low 32 bits

STR R<sub>5</sub>, [R<sub>0</sub>, #4] speicher high 32 bits

BX LR

Addr R<sub>1</sub>, R<sub>2</sub>  $(\text{int} b_4 \dots a \dots (\text{int} b_4 \dots t) b) \Rightarrow a(R_1) \rightarrow (a+4 \times b)/2$  SS 18

LDM R<sub>1</sub>, {R<sub>3</sub>, R<sub>4</sub>}

LDRH R<sub>5</sub>, [R<sub>2</sub>]  $R_5 = b$  (half word)

MOV R<sub>5</sub>, R<sub>5</sub>, LSL #2  $R_5 = b * 4$

ADDS R<sub>3</sub>, R<sub>3</sub>, R<sub>5</sub>

ADC R<sub>4</sub>, R<sub>4</sub>, #0

MOV R<sub>3</sub>, R<sub>3</sub>, LSR #1

ORR R<sub>3</sub>, R<sub>3</sub>, R<sub>4</sub>, LSL #31

MOV R<sub>4</sub>, R<sub>4</sub>, LSR #1

STM R<sub>1</sub>, {R<sub>3</sub>, R<sub>4</sub>}

$(\text{int} b_4 \dots t) a \dots (\text{int} b_4 \dots t) b \Rightarrow a \leftarrow (a+63 \times b)/2$

LDM R<sub>1</sub>, {R<sub>3</sub>, R<sub>4</sub>}

LDRH R<sub>5</sub>, [R<sub>2</sub>]  $R_5 = b_3$

MOV R<sub>6</sub>, #63

MUL R<sub>5</sub>, R<sub>5</sub>, R<sub>6</sub>  $R_5 = b_3 * b$

MOV R<sub>6</sub>, #0  $R_6 = \text{high 32 bit von } b$

ADDS R<sub>3</sub>, R<sub>3</sub>, R<sub>5</sub>  $R_3 = a\text{-low} + (b_3 \cdot b)\text{-low. mit Carry}$

ADC R<sub>4</sub>, R<sub>4</sub>, R<sub>6</sub>  $R_4 = a\text{-high} + (b_3 \cdot b)\text{-high + Carry}$

Hinweise:

div 2 ADDMI ... #1, ASR 1

div 4 ADDMI ... #3, ASR 2

div 8 ADDMI ... #7, ASR 3

$(\text{int} b_4 \dots a \dots (\text{int} b_4 \dots t) b) \Rightarrow a(R_1) \rightarrow (a+b)/2$

LDM R<sub>1</sub>, {R<sub>3</sub>, R<sub>4</sub>}

LDRSH R<sub>5</sub>, [R<sub>2</sub>]  $R_5 = b$  (Vorzeichen)

MOV R<sub>6</sub>, R<sub>5</sub>, ASR #31 Erweitere zu 32 bit

ADDS R<sub>3</sub>, R<sub>3</sub>, R<sub>5</sub> Erzeuge hohes 32 bit von b

ADC R<sub>4</sub>, R<sub>4</sub>, R<sub>6</sub>

MOV R<sub>3</sub>, R<sub>3</sub>, ASR #1

ORR R<sub>3</sub>, R<sub>3</sub>, R<sub>4</sub>, LSL #31

MOV R<sub>4</sub>, R<sub>4</sub>, ASR #1

STM R<sub>1</sub>, {R<sub>3</sub>, R<sub>4</sub>}

(arithmetische)

$R_3 = R_3 \gg 1$

MOV R<sub>3</sub>, R<sub>3</sub>, ASR #1

ORR R<sub>3</sub>, R<sub>3</sub>, R<sub>4</sub>, LSL #31

MOV R<sub>4</sub>, R<sub>4</sub>, ASR #1

STM R<sub>1</sub>, {R<sub>3</sub>, R<sub>4</sub>}

In R<sub>0</sub> eine 12 bit Zahl ( $0 \dots 2^{12}-1$ ), ab Bitposition 4  $\Rightarrow a(R) \leftarrow a:10$  a in 32 bits

LSR R<sub>2</sub> R<sub>0</sub> #4 Recht schieben (R<sub>2</sub>=R<sub>0</sub> >> 4 Magic Number:  
 AND R<sub>2</sub> R<sub>2</sub> #0FFF bits von 0 bis 11

MOV R<sub>1</sub> #0 R<sub>1</sub> = Quotient

loop\_div10:

CMP R<sub>2</sub> #10 fertig, wenn a<10  
 BLT done\_div10

SUB R<sub>2</sub> R<sub>2</sub> #10 a -= 10

ADD R<sub>1</sub> R<sub>1</sub> #1 Quotient + 1

B loop\_div10

done\_div10: R<sub>1</sub> = a : 10

Zur Implementierung für den ARM7 braucht man nur eine normale Multiplikation mit 32-Bit-Ergebnis.

#### Assembler

#092

```

1 LDR   R1, =0xB6DB6DB7 ; R0 (unsigned) sei durch 7 zu dividieren
2 LDR   R1, =0xB6DB6DB7 ; R1 ← 0xB6DB6DB7
3 MUL  R3, R0, R1        ; R3 ← R0 · R1 = R0 · 0xB6DB6DB7 = R0 ÷ 7 (mod 232)
4 MOV  R3, R3, LSR #1    ; R3 ← R3 ÷ 2 = R0 ÷ 10

```

Die Division durch 10 wird durch die Division durch 5 und anschließender Division durch 2 realisiert.

#### Assembler

#093

```

1 LDR   R1, =0xCCCCCCCC ; R0 (unsigned) sei durch 10 zu dividieren
2 LDR   R1, =0xCCCCCCCC ; R1 ← 0xCCCCCCCC
3 MUL  R3, R0, R1        ; R3 ← R0 · R1 = R0 · 0xCCCCCCCC = R0 ÷ 5 (mod 232)
4 MOV  R3, R3, LSR #1    ; R3 ← R3 ÷ 2 = R0 ÷ 10

```

K	I
3	0xAAAAAAAB
5	0xCCCCCCCC
7	0xB6DB6DB7
9	0x38E38E39
11	0xBA2E8BA3
13	0xC4EC4EC5
15	0xEEEEEEEF
17	0xF0F0F0F1
19	0x286BCA1B
21	0x3CF3CF3D
23	0xE9BD37A7
25	0xC28F5C29
255	0xFEFEFEFF
65535	0xFFFFFFF

Tab. 19: Modulare multiplikative Inverse I zur 32-Bit-Division mit Konstanten K für den ARM7-Prozessor. Quelle: „Hacker's Delight“ von H. S. Warren, Addison-Wesley (2008)

64 bit Wert x in R<sub>0</sub> (low), R<sub>1</sub> (high)  $\Rightarrow a \leftarrow (x * 128) \bmod 2^{64} \rightarrow$  Ergebniss in 64 Bits

MOV R<sub>2</sub> R<sub>0</sub> LSL #7 R<sub>2</sub> = x.low + 2<sup>7</sup>

MOV R<sub>4</sub> R<sub>0</sub> LSR #25 R<sub>4</sub> = extrahiert high 7 bit von x.low

MOV R<sub>3</sub> R<sub>1</sub> LSL #7 R<sub>3</sub> = x.high + 2<sup>7</sup>

ORR R<sub>3</sub> R<sub>3</sub> R<sub>4</sub> R<sub>3</sub> = Die aus R<sub>0</sub> verschobenen 7 bit zu R<sub>3</sub> hinzufügen (neue high 31 bit)

MOV R<sub>0</sub> R<sub>2</sub>

MOV R<sub>1</sub> R<sub>3</sub>

x  $\leftarrow (x \cdot 2^{57}) \bmod 2^{64}$   $x \cdot 2^{57} = 2^8 \cdot x + x$

MOV R<sub>2</sub> R<sub>0</sub> LSL #8

MOV R<sub>4</sub> R<sub>0</sub> LSR #24

MOV R<sub>3</sub> R<sub>1</sub> LSL #8

ORR R<sub>3</sub> R<sub>2</sub> R<sub>4</sub>

ADDS R<sub>0</sub> R<sub>2</sub> R<sub>0</sub>

ADC R<sub>1</sub> R<sub>3</sub> R<sub>1</sub>

Schreiben Sie ein Programm, das für die ARMv6-Architektur eine Wort-variable (Adresse in R0 gegeben) thread-sicher modifiziert. Es soll in dieser Variable

- Bit 4 gelöscht werden, wenn Bit 0 den Wert 1 hat
- Bit 5 gesetzt werden, wenn Bit 1 und Bit 2 den Wert 0 haben
- Bit 6 gewechselt (0 auf 1 und umgekehrt) werden
- der 7-Bit-Zähler (Wertebereich 0 ... 127) ab der Bitposition 7 mit Prüfung der Wertebereichsgrenzen dekrementiert (nicht unter 0) und
- der 18-Bit-Zähler (Wertebereich -214 ... 214 - 1) ab der Bitposition 14 ohne Prüfung der Wertebereichsgrenzen inkrementiert werden.

Die Bits auf den Positionen 0 bis 3 dürfen nicht verändert werden.

STMFD SP!, {R0-R3, LR}

Retry:

LDR EX R1 [R0]  
AND R2 R1 #0XF

Exklusiv laden 32 bit R0 in R1,  
speichern Bit 0-3 in R2

TST R1 #0X1

BIC NE R1 R1 #0x10

wenn bit 0 = 1 lösche Bit 4

AND R3 R1 #0x6

Extrahiere bit 1 and 2 in R3

CMP R3 #0

Vergleiche bit 1 and 2 mit 0

ORR EQ R1 R1 #0x20

Wenn bit 1 and 2 = 0, setze Bit F = 1

EOR R1 R1 #0x40

Invertiere Bit 6

(unsigned Zahl)

AND R3 R1 #0X3F80

Extrahiere bit 7-13 in R3 (mask 0x3F80)

MUV R3 R3 LSR #7

R3 > 7, erhalte bit 7-13

CMP R3 #0

Wenn 0, nicht dekrementieren.

BZR R1 R1 skip-decrement

SUB R3 R3 #1

MOV R3 R3 LSL #7

BIC R1 R1 #0X3F80

Lösche ursprüngliche Bit 7-13

ORR R1 R1 R3

Schreibe neuen Wert.

skip-decrement:

ADD R1 R1 #0X4000

Überlauf wird nicht geprüft, direkt addiere

$2^{16} = 0x4000$

BIC R1 R1 #0XF

Löschen Bit 0-3

ORR R1 R1 R2

wieder herstellen des ursprünglichen Wertes.

STREX R3 R1 [R0]

Schreibe R1 nach [R0], Ergebnis in R3

CMP R3 #0

R3 = 0 erfolgreich, R3 = 1, nicht

BNE retry

LDMDF SP! {R0-R3, PC}

- Bit 4 gesetzt werden, wenn die Bits 0 und 1 jeweils gleich 1 sind
- Bit 5 gelöscht werden, wenn die Bits 2 und 3 gleich 0 sind
- Bit 6 gewechselt (0 auf 1 und umgekehrt) werden
- Der 9-Bit-Wert auf den Bitpositionen 7 bis 15 dividiert durch 10

Die übrigen Bits dürfen nicht angefasst werden!

STMFD SP!, {R0, R1, R2, R3, R4, R5, R6, R8, LR}; Register sichern (inkl. Rücksprungadresse)

retry:

LDREXH R1, [R0] ; Lade 16-Bit-Wert atomar aus Adresse in R0 nach R1

; Wenn Bit 0 und 1 gleich 1 → Bit 4 setzen

AND R2, R1, #0x3 ; Maske: Bit 0 und 1 extrahieren

CMP R2, #0x3 ; Vergleiche mit 0b11

ORREQ R1, R1, #0x10 ; Wenn gleich → Bit 4 (0x10) setzen

; Wenn Bit 2 und 3 gleich 0 → Bit 5 löschen

ANDS R2, R1, #0xc ; Maske: Bit 2 und 3 extrahieren

BICEQ R1, R1, #0x20 ; Wenn Ergebnis = 0 → Bit 5 (0x20) löschen

; Bit 6 invertieren (0 <-> 1)

EOR R1, R1, #0x40 ; Bit 6 (0x40) umschalten

; Bits 7-15 extrahieren und durch 10 teilen

LDR R6, =0xFF80 ; Maske für Bits 7 bis 15

AND R2, R1, R6 ; Extrahiere Bits 7-15 in R2

LDR R3, =0XCCCCCCC ; Magic Number für Division durch 10

UMULL R4, R5, R2, R3 ; 64-Bit-Multiplikation: R5 enthält High-Teil

MOV R2, R5, LSR #3 ; Division durch 10 (Rechtsverschiebung)

BIC R1, R1, R6 ; Alte Bits 7-15 löschen

ORRR R1, R1, R2 ; Ergebnis der Division in Bits 7-15 einfügen

; versuche, neuen Wert zurückzuschreiben (atomar)

STREXH R8, R1, [R0] ; Schreibe R1 in [R0], Ergebnis → R8

CMP R8, #0 ; War der Schreibvorgang erfolgreich?

BNE retry ; Wenn nicht → wiederhole gesamte Operation

- Bit 8 gesetzt werden, wenn die Bits 0 und 1 jeweils gleich 0 sind
  - Bit 9 gelöscht werden, wenn die Bits 2 und 3 gleich 1 sind
  - Bit 10 gewechselt werden, wenn die Bits 4 und 5 den gleichen Wert haben
  - Der 5-Bit-Zähler auf den Bitpositionen 11 bis 15 inkrementiert (modulo 32) werden
- Die Bits 0 bis 7 sowie 16 bis 31 dürfen nicht verändert werden!

STMF D SP!, {R0-R4, R6, R7, R8, LR} ; Register sichern

retry:

LDREX R1, [R0] ; Lade 32-bit Wert atomar

; ----- Bedingung 1: Bit 0 und 1 == 0 → Setze Bit 8 -----

ANDS R2, R1, #0x3 ; Maske: Bit 0 & 1

ORREQ R1, R1, #0x100 ; Wenn beide 0 → Bit 8 setzen

; ----- Bedingung 2: Bit 2 und 3 == 1 → Lösche Bit 9 -----

AND R2, R1, #0xc ; Maske: Bit 2 & 3

CMP R2, #0xc ; Sind beide 1?

BICEQ R1, R1, #0x200 ; Dann Bit 9 löschen

; ----- Bedingung 3: Bit 4 == Bit 5 → Wechsle Bit 10 -----

EOR R2, R1, R1, LSR #1 ; Wenn R2 = 0, bit4 = bit5

TSTR R2, #0x10 ; Prüfe Bit 4

EOREQ R1, R1, #0x400 ; Wenn gleich → Bit 10 invertieren

; ----- Bedingung 4: Inkrementiere Bits 11-15 (mod 32) -----

MOV R2, R1, LSR #11 ; R2 = Bits 11-31

AND R2, R2, #0x1F ; Extrahiere Bits 11-15

ADD R2, R2, #1 ; +1

AND R2, R2, #0x1F ; Modulo 32 (limitiert auf 5 bit)

BIC R1, R1, #0xF800 ; Lösche alte Bits 11-15 im R1

ORR R1, R1, R2, LSL #11 ; Füge neue Bits 11-15 ein

STREX R3, R1, [R0] ; Versuche zu speichern

CMP R3, #0

BNE retry ; Wenn fehlgeschlagen → wiederholen

$$S = \sum_{i=0}^n u_i v_{n-i}$$

R2       $u_i$  : 8 bit Zahl ( $-2^7 \dots 2^7$ )  
 R3       $v_i$  : 16 bit Zahl (0 ...  $2^{16}-1$ )  
 Output    R1 (high) R0 (low)      64 bits  
 $i = 0 \dots n$

; Eingabe:

- ; R2 - Zeiger auf das  $u_i$ -Array (8-Bit vorzeichenbehaftet)
- ; R3 - Zeiger auf das  $v_i$ -Array (16-Bit vorzeichenlos)
- ; R4 - Wert von n ( $n \geq 1$ )

; Ausgabe:

- ; R0 - Niedrigwertige 32 Bits des Ergebnisses
- ; R1 - Hochwertige 32 Bits des Ergebnisses

calculate\_S:

STMFD SP!, {R4-R8, LR} ; Register sichern, inkl. LR für 8-Byte-Alignierung

```

MOV R0, #0          ; Ergebnis (niedrig 32 Bit) initialisieren mit 0
MOV R1, #0          ; Ergebnis (hoch 32 Bit) initialisieren mit 0
ADD R4, R4, #1      ; Schleifenanzahl = n + 1 (i = 0 bis n)

```

; Berechne Endadresse des v-Arrays:  $R8 = R3 + 2 * (n + 1)$

```

MOV R8, R4          ; Kopiere n+1 nach R8
ADD R8, R3, R8, LSL #1 ; R8 = R3 + 2 * (n + 1)

```

loop:

LDRSB R5, [R2], #1 ; Lade vorzeichenbehaftetes 8-Bit  $u_i$  nach R5, R2 inkrementieren

; Adresse von  $v_{[n-i]}$  berechnen:  $R6 = R8 - 2 * (i + 1)$

```

SUB R6, R8, R4, LSL #1 ; R6 = R8 - 2 * R4
LDRH R6, [R6]           ; Lade vorzeichenloses 16-Bit  $v_{[n-i]}$  nach R6

```

; Vorzeichenbehaftete 32-Bit Multiplikation: Ergebnis in R7 (low) und R8 (high)

SMULL R7, R8, R5, R6 ; R7:R8 = R5 \* R6 (64 Bit)

; 64-Bit Addition zum Ergebnis (R1:R0)

```

ADDS R0, R0, R7        ; Addiere Low-Teil, setze Carry-Flag
ADC R1, R1, R8         ; Addiere High-Teil mit Übertrag

```

SUBS R4, R4, #1 ; Dekrementiere Schleifenzähler

BGT loop ; Wiederhole, solange R4 > 0

LDMFD SP!, {R4-R8, PC} ; Wiederherstellung der Register und Rückkehr

$S = \sum_{i=0}^n a_i b_{n-i}$   
 R<sub>2</sub> a 16 bit Zahl ( $0 \dots 2^{16}-1$ )  
 R<sub>3</sub> b 8 bit Zahl ( $-2^7 \dots 2^7$ )  
 Output R<sub>0</sub> 32 bits  
 $i = 0 \dots n$

1. MOV R5, #0 ; Initialisiere Akkumulator (R5 = 0)
2. ADD R1, R1, R2 ; b-Array Pointer + N (weil b[N-i] verwendet wird)
- loop:
3. LDRH R<sub>3</sub>, [R<sub>0</sub>], #2 ; Lade a<sub>i</sub> (16-Bit Wert, Post-Inkrement um 2)
4. LDRSB R<sub>4</sub>, [R<sub>1</sub>], #-1 ; Lade b<sub>{N-i}</sub> (8-Bit vorzeichenbehaftet, Post-Dekrement um 1)
5. MLA R<sub>5</sub>, R<sub>3</sub>, R<sub>4</sub>, R<sub>5</sub> ; Multipliziere und addiere: R<sub>5</sub> = R<sub>5</sub> + (R<sub>3</sub> \* R<sub>4</sub>)
6. SUBS R<sub>2</sub>, R<sub>2</sub>, #1 ; Dekrementiere Zähler (N = 1), setze Statusflags
7. BNE loop ; Falls Z-Flag nicht gesetzt (N ≠ 0), springe zu loop
8. MOV R<sub>0</sub>, R<sub>5</sub> ; Speichere Endergebnis in R<sub>0</sub>

-----	-----	-----
`MOV R5, #0`	`E3A05000`	
`ADD R1, R1, R2`	`E0811002`	
`LDRH R3, [R0], #2`	`E0D030B2`	
`LDRSB R4, [R1], #-1`	`E1D140D1`	
`MLA R5, R3, R4, R5`	`E0234593`	
`SUBS R2, R2, #1`	`E2522001`	
`BNE loop`	`1AFFFFFFA`	
`MOV R0, R5`	`E1A00005`	

### Instruktions Wörter hexadezimalen /zyklen

SBRS R<sub>1</sub> R<sub>2</sub> R<sub>3</sub> LSR R<sub>4</sub> Zyklen: 2 Dateium. Rd! = PC, mit BS

LDR R<sub>0</sub> [R<sub>1</sub>, R<sub>2</sub>]!  $\rightarrow$  LDR Rd! = PC

LDRSH R<sub>0</sub> [R<sub>1</sub>, 0x200] X 0x100 > max offset 8bit

MLAEQ R<sub>0</sub> R<sub>1</sub> R<sub>2</sub> R<sub>3</sub> 3-6 MCA

ADCSLSS PC R<sub>0</sub> #4 3 Datenum. RC = PC. ohne BS weil kein rotatе\_im

Z	<table border="1"> <thead> <tr> <th>31..28</th><th>27</th><th>26</th><th>25</th><th>24 .. 21</th><th>20</th><th>19 .. 16</th><th>15 .. 12</th><th>11 .. 8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3 .. 0</th></tr> </thead> <tbody> <tr> <td>AL</td><td>I</td><td>Sub</td><td>S</td><td>Rn(R<sub>2</sub>)</td><td>Rd(R<sub>0</sub>)</td><td>RS(R<sub>4</sub>)</td><td>LSR</td><td>Rm(R<sub>3</sub>)</td><td></td><td></td><td></td><td></td><td></td></tr> <tr> <td>1110 / 0 0 0</td><td>0 0 10</td><td>1 /</td><td>0010 /</td><td>0001 /</td><td>0100 /</td><td>0011 /</td><td>0011 /</td><td>0011 /</td><td></td><td></td><td></td><td></td><td></td></tr> <tr> <td>E</td><td>0</td><td>5</td><td></td><td>2</td><td>1</td><td>4</td><td>3</td><td>3</td><td></td><td></td><td></td><td></td><td></td></tr> </tbody> </table>	31..28	27	26	25	24 .. 21	20	19 .. 16	15 .. 12	11 .. 8	7	6	5	4	3 .. 0	AL	I	Sub	S	Rn(R <sub>2</sub> )	Rd(R <sub>0</sub> )	RS(R <sub>4</sub> )	LSR	Rm(R <sub>3</sub> )						1110 / 0 0 0	0 0 10	1 /	0010 /	0001 /	0100 /	0011 /	0011 /	0011 /						E	0	5		2	1	4	3	3															
31..28	27	26	25	24 .. 21	20	19 .. 16	15 .. 12	11 .. 8	7	6	5	4	3 .. 0																																																						
AL	I	Sub	S	Rn(R <sub>2</sub> )	Rd(R <sub>0</sub> )	RS(R <sub>4</sub> )	LSR	Rm(R <sub>3</sub> )																																																											
1110 / 0 0 0	0 0 10	1 /	0010 /	0001 /	0100 /	0011 /	0011 /	0011 /																																																											
E	0	5		2	1	4	3	3																																																											
3	<table border="1"> <thead> <tr> <th>31..28</th> <th>27</th> <th>26</th> <th>25</th> <th>24</th> <th>23</th> <th>22</th> <th>21</th> <th>20</th> <th>19 .. 16</th> <th>15 .. 12</th> <th>11 .. 8</th> <th>7</th> <th>6</th> <th>5</th> <th>4</th> <th>3 .. 0</th> </tr> </thead> <tbody> <tr> <td>AL</td> <td>I</td> <td>P</td> <td>V</td> <td>B</td> <td>W</td> <td>L</td> <td>Rn(R<sub>1</sub>)</td> <td>Rd(R<sub>0</sub>)</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>1110 / 0 1</td> <td>1</td> <td>1</td> <td>1</td> <td>0</td> <td>1</td> <td>1 /</td> <td>0001 /</td> <td>0000 /</td> </tr> <tr> <td>E</td> <td>7</td> <td>B</td> <td></td> <td></td> <td></td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>2</td> </tr> </tbody> </table>	31..28	27	26	25	24	23	22	21	20	19 .. 16	15 .. 12	11 .. 8	7	6	5	4	3 .. 0	AL	I	P	V	B	W	L	Rn(R <sub>1</sub> )	Rd(R <sub>0</sub> )									1110 / 0 1	1	1	1	0	1	1 /	0001 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /	E	7	B				1	0	0	0	0	0	0	0	0	2
31..28	27	26	25	24	23	22	21	20	19 .. 16	15 .. 12	11 .. 8	7	6	5	4	3 .. 0																																																			
AL	I	P	V	B	W	L	Rn(R <sub>1</sub> )	Rd(R <sub>0</sub> )																																																											
1110 / 0 1	1	1	1	0	1	1 /	0001 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /																																																				
E	7	B				1	0	0	0	0	0	0	0	0	2																																																				

3-6	<table border="1"> <thead> <tr> <th>31..28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19 .. 16</th><th>15 .. 12</th><th>11 .. 8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3 .. 0</th></tr> </thead> <tbody> <tr> <td>ALL</td><td>I</td><td>P</td><td>U</td><td>B</td><td>W</td><td>L</td><td>Rn</td><td>Rd</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr> <td>1110 / 0 0</td><td>0 1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1 /</td><td>0001 /</td><td>0000 /</td><td>0000 /</td><td>0000 /</td><td>0000 /</td><td>0000 /</td><td>0000 /</td><td>0000 /</td><td>0000 /</td></tr> <tr> <td>E</td><td>1</td><td>D</td><td></td><td></td><td></td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>F</td><td>0</td><td></td><td></td></tr> </tbody> </table>	31..28	27	26	25	24	23	22	21	20	19 .. 16	15 .. 12	11 .. 8	7	6	5	4	3 .. 0	ALL	I	P	U	B	W	L	Rn	Rd									1110 / 0 0	0 1	1	1	0	1	1 /	0001 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /	E	1	D				1	0	0	0	0	0	F	0			LDRSH R <sub>0</sub> , [R <sub>1</sub> , 0x 00]
31..28	27	26	25	24	23	22	21	20	19 .. 16	15 .. 12	11 .. 8	7	6	5	4	3 .. 0																																																				
ALL	I	P	U	B	W	L	Rn	Rd																																																												
1110 / 0 0	0 1	1	1	0	1	1 /	0001 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /																																																					
E	1	D				1	0	0	0	0	0	F	0																																																							

3	<table border="1"> <thead> <tr> <th>31..28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19 .. 16</th><th>15 .. 12</th><th>11 .. 8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3 .. 0</th></tr> </thead> <tbody> <tr> <td>PC (R15)</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr> <td>0010 / 0 0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1 /</td><td>0000 /</td><td>1111 /</td><td>0000 /</td><td>0000 /</td><td>0000 /</td><td>0000 /</td><td>0000 /</td><td>0000 /</td><td>0000 /</td><td></td></tr> <tr> <td>E</td><td>2</td><td>Z</td><td>B</td><td></td><td></td><td>0</td><td>F</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>4</td><td></td></tr> </tbody> </table>	31..28	27	26	25	24	23	22	21	20	19 .. 16	15 .. 12	11 .. 8	7	6	5	4	3 .. 0	PC (R15)																	0010 / 0 0	1	0	1	0	1	1 /	0000 /	1111 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /		E	2	Z	B			0	F	0	0	0	0	0	0	0	4	
31..28	27	26	25	24	23	22	21	20	19 .. 16	15 .. 12	11 .. 8	7	6	5	4	3 .. 0																																																					
PC (R15)																																																																					
0010 / 0 0	1	0	1	0	1	1 /	0000 /	1111 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /	0000 /																																																						
E	2	Z	B			0	F	0	0	0	0	0	0	0	4																																																						

Sei dir Formel

$$f = \sum_{i=0}^{N-1} a_i * b_{N-i-1}$$

gegeben, wobei die Koeffizienten  $a_i$  durch ganze 8 Bit Zahl ( $0 \dots 2^8 - 1$ ) und die Koeffizienten  $b_i$  durch ganze 16 Bit Zahlen ( $-2^{15} \dots -2^{15} - 1$ ) dargestellt werden ( $i = 0, 1 \dots N-1; N \leq 99$ ).

Schreiben Sie ein Programm zur Berechnung von  $f$ .  $R0$  und  $R1$  zeigen jeweils auf ein Feld mit den Koeffizienten  $a_i$  und  $b_i$ ,  $N$  wird durch  $R2$  angegeben. Es gilt  $N > 1$ . Sie dürfen davon ausgehen, dass  $-2^{31} \leq f < 2^{31}$  gilt. Rückgabe von  $f$  in Register  $R0$ .

**R0: a: 8 Bit (+)**  
**R1: b: 16 Bit (+-)**  
**R2: N**

```
1. MOV      R5, #0          ; R5 clearn
2. ADD      R1, R1, R2, LSL#1 ; N*2 weil Zahlen in b 2 Byte
                                ; von hinten anfangen
loop
3. LDRB    R3, [R0], #1      ; 1 Byte von a in R3 laden
4. LDRSH   R4, [R1, #-2]!   ; signed Halfword (2 Byte)
                                ; von b in R4 laden
5. MLA     R5, R3, R4, R5  ; R5 = R5 + ai * bN-i-1
6. SUBS   R2, R2, #1        ; N dekrementieren
7. BNE    loop              ; wenn > 0: wiederholen
8. MOV     R0, R5            ; Ergebnis in R0
```

MLA Rd Rn Rm Ra       $Rd = (Rn * Rm) + Ra$

Schreiben Sie ein Unterprogramm, das einen Speicherinhalt mit Nullen überschreibt. Das SRAM-Speicher-Interface verfügt über einen 32 Bit breiten Datenbus ohne Cache. Die Adresse des auszuführenden Speicherbereichs wird durch R0 und die Anzahl der Bytes ( $\geq 7$ ) in R1 angegeben.

Die Funktion soll aus Gründen der Laufzeitminimierung, wenn möglich, die volle Busbandbreite ausnutzen und zu allen Architekturen ab ARMv5 kompatibel sein.

Schreiben Sie ein Unterprogramm, das einen Speicherinhalt kopiert. Das SRAM-Speicher-Interface verfügt über einen 32-Bit breiten Datenbus ohne Cache.

Die Adresse des zu kopierenden Bereiches wird durch R0, die Zieladresse durch R1 und die Anzahl Bytes ( $\geq 7$ ) in R2 angegeben. Die Speicherbereiche überlappen sich nicht.

Die Funktion soll aus Gründen der Laufzeitoptimierung, wenn möglich, die volle Busbandbreite ausnutzen und zu allen Architekturen ab ARMv5 kompatibel sein.

R0 = Startadresse  
R1 = Zieladresse  
R2 = Anzahl zu kopierender Bytes

```
1. MOV R4, R2, LSR#2 ; R0: Adresse des Speicherbereichs
   ; R1: Anzahl der Bytes ( $\geq 7$ )
   ; Anzahl Bytes durch 4
   ; R4 cldaren für Nullen

loop
  1. MOV R2, R1, LSR#2 ; Anzahl Bytes durch 4
   ; R2, R1, LSR#2 ; Anzahl Bytes durch 4
   ; R4, #0          ; R4 cldaren für Nullen

  2. TST R1, #0x2    ; ob Anzahl Bytes % 2 > 0
   ; von R4 1 Byte in R0 schreiben
   ; erhöhe R1 um 1

  3. STR R4, [R0], #4 ; R4 in R0, erhöhe R0 um 4
   ; Anzahl Wörter da sind
   ; Anzahl Wörter dekrementieren
   ; solange != 0

  4. SUBS R2, R2, #1 ; von R3 4 Bytes in R1
   ; schreiben, erhöhe R1 um 4
   ; Anzahl Wörter dekrementieren
   ; solange != 0

  5. BNE loop       ; von R4 2 Byte in R0 schreiben
   ; erhöhe R1 um 2

  6. TST R2, #2     ; ob Anzahl Bytes % 2 > 0
   ; von R3 2 Bytes in R0 schreiben
   ; erhöhe R1 um 2

  7. LDRHNE R3, [R0], #2 ; R1 durch 4 9:1001 2x-> 0010
   ; R1 9:1001, R2<-2x: 1000
   ; 1001-1000=0001

  8. STRHNE R3, [R1], #2 ; von R3 1 Byte in R1 schreiben
   ; erhöhe R1 um 1

  9. TST R2, #1     ; ob Anzahl Bytes % 1 > 0
   ; 1 Byte aus R0 in R3,
   ; erhöhe R0 um 1
   ; von R3 1 Byte in R1 schreiben
   ; erhöhe R1 um 1

10. LDRHNE R3, [R0], #1 ; R1 9:1001, R2<-2x: 1000
   ; 1001-1000=0001
```

Hinweise:

2.  
-----  
; R1 durch 4 9:1001 2x-> 0010  
; R1 9:1001, R2<-2x: 1000  
; 1001-1000=0001

## # ARM 指令执行结果分析

给定初始寄存器值和标志位状态：

- R0 = 0x00000000
- R1 = 0xAA0AA001
- R2 = 0x88055002
- 标志位：C=1, N=0, Z=0, V=0

我们依次分析每条指令执行后的R0值：

## (a) EORS R0, R1, R2

\*\*异或操作\*\*:  $R0 = R1 \text{ XOR } R2$

$0xAA0AA001 \text{ XOR } 0x88055002 = 0x220FF003$

同时会更新标志位 (N和Z)

\*\*结果\*\*:  $R0 = 0x220FF003$

## (b) ADDEQ R0, R1, R2, LSR R1

\*\*条件执行\*\*: EQ 表示 Z=1 时才执行，当前 Z=0，所以\*\*不执行\*\*

\*\*结果\*\*: R0 保持原值 = 0x00000000

## (c) ADC R0, R1, R2

\*\*带进位加法\*\*:  $R0 = R1 + R2 + C$

$0xAA0AA001 + 0x88055002 + 1 = 0x320FF004$

(注意进位可能会影响标志位)

\*\*结果\*\*:  $R0 = 0x320FF004$

## (d) AND R0, R1, R2

\*\*逻辑与操作\*\*:  $R0 = R1 \text{ AND } R2$

$0xAA0AA001 \text{ AND } 0x88055002 = 0x88050000$

(不会影响标志位)

\*\*结果\*\*:  $R0 = 0x88050000$

## (e) SUB R0, R1, R2, ASR #8

\*\*算术右移减法\*\*:

先计算  $R2 \text{ ASR } #8 = 0xFF880550$

然后  $R0 = R1 - (R2 \text{ ASR } #8) = 0xAA0AA001 - 0xFF880550 = 0xAA824AB1$

(会设置标志位)

\*\*结果\*\*:  $R0 = 0xAA824AB1$

## (f) SUBNE R0, R1, R2, ASR #8

\*\*条件减法\*\*: NE 表示 Z=0 时才执行，当前 Z=0，所以执行

同 (e) 的计算:  $R0 = 0xAA824AB1$

\*\*结果\*\*:  $R0 = 0xAA824AB1$

## (g) TST R0, R1, R2

\*\*测试指令\*\*: 执行  $R1 \text{ AND } R2$  但不存储结果，只更新标志位

$R1 \text{ AND } R2 = 0x88050000$