



**h\_da**

HOCHSCHULE DARMSTADT  
UNIVERSITY OF APPLIED SCIENCES

**fbi**  
FACHBEREICH INFORMATIK

RECHNERARCHITEKTUR

WS2013

Termin 3

Arithmetische und logische Operationen

Name, Vorname	Matrikelnummer	Anmerkungen
Datum	Raster (z.B. Mi3x)	Testat/Datum

Legende: V:Vorbereitung, D: Durchführung, P: Protokoll/Dokumentation, T: Testat

**Ziele:**

Verständnis für arithmetische und logische Operationen. Ziel ist die Implementierung mit möglichst geringer Codegröße sowie das Erlernen und Festigen des Umgangs mit einer Entwicklungsumgebung.

**Vorbereitung**

Arbeiten Sie sich in die datenverarbeitenden Befehle des ARM-Prozessors ein:

Instruktion	Bedeutung
AND	Rd = Op1 AND Op2
EOR	Rd = Op1 EOR Op2
SUB	Rd = Op1 - Op2
RSB	Rd = Op2 - Op1
ADD	Rd = Op1 + Op2
ADC	Rd = Op1 + Op2 + Carry
SBC	Rd = Op1 - Op2 - Carry
RSC	Rd = Op2 - Op1 - Carry
TST	setzt Condition Codes bzgl. Op1 And Op2
TEQ	setzt Condition Codes bzgl. Op1 EOR Op2
CMP	setzt Condition Codes bzgl. Op1 - Op2
CMN	setzt Condition Codes bzgl. Op1 + Op2
ORR	Rd = Op1 ORR Op2
MOV	Rd = Op2
BIC	Rd = Op1 AND NOT Op2
MVN	Rd = NOT Op2 (Einerkomplement)

**Aufgabe 1:**

Was leisten die folgenden beiden Befehle?

MOV R0, R1, LSR #2 \_\_\_\_\_

ADD R0, R1, R1, LSL #3 \_\_\_\_\_

**Aufgabe 2:**

Überlegen Sie sich, mit welchen Befehlen Sie die einzelnen Flags (NZCV) gesetzt bekommen.

Zum Beispiel setzt der Befehl: ADDS r0, r0, r0 nur das Zero-Flag wenn R0=0

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

### Aufgabe 3:

Füllen Sie die unten stehende Tabelle aus.

Die Register haben folgende Werte:

R0 = 0xAABBCCDD

R1 = 0xFFBBFFBB

R2 = 0xFFFFFFFF

R3 = zum Beispiel Ihre Matrikelnummer (rechtsbündig, Hexadezimalzahl)

R4 = 0x3

R5 = 0x2

R6 = 0x7ffffff

R7 = 0x80000000

Instruktion	R9 (hexadez.)	Zusatzfrage	Antwort
ANDS R9, R0, R3		Wie werden die Flags N, Z, C, V gesetzt?	_ _ _ _
EOR R9, R3, R3		Gilt das Ergebnis für jeden Wert in R3?	Ja/Nein
SUBS R9, R7, #3		Wie werden die Flags N, Z, C, V gesetzt?	_ _ _ _
RSBS R9, R5, #3		Wie werden die Flags N, Z, C, V gesetzt?	_ _ _ _
ADDS R9, R4, #12		Wie werden die Flags N, Z, C, V gesetzt?	_ _ _ _
ADDS R9, R6, R4		Wie werden die Flags N, Z, C, V gesetzt?	_ _ _ _
TST R4, #1	-	Wie werden die Flags N, Z, C, V gesetzt?	_ _ _ _
TEQ R4, R4	-	Wie werden die Flags N, Z, C, V gesetzt?	_ _ _ _
CMP R5, R4	-	Wie werden die Flags N, Z, C, V gesetzt?	_ _ _ _
CMN R2, R5	-	Wie werden die Flags N, Z, C, V gesetzt?	_ _ _ _
ORR R9, R0, R3			
MOV R9, #126			
BIC R9, R0, R1			
BIC R9, R2, #15			
MVN R9, R1			

### Arbeitsverzeichnis:

Melden Sie sich mit Ihre Fachbereichszugangsdaten (istxxxxx/Passwort) an. Kopieren Sie sich vom zur Verfügung stehende Netzwerklaufwerk (/mnt/Originale/ra) das Verzeichnis raWS2013 in Ihr Arbeitsverzeichnis. Wechseln Sie in das Verzeichnis ~/raWS2013/Termin3/. Starten Sie den *snavigator* im Arbeitsverzeichnis und legen sich ein neues Projekt an.

### Aufgabe 4:

Es sind in den Registern R0 bis R3 Werte gegeben die Sie auf vier verschiedene Arten jeweils mit -1 multiplizieren (Vorzeichen umkehren, 2K-Wandlung) sollen. Testen und dokumentieren Sie Ihr Programm.

### Aufgabe 5:

Überprüfen Sie Ihre Vorbereitung, wenn noch nicht geschehen, durch Debuggen entsprechender gegebener oder evtl. selbst geschriebener Programme.

### **Zusatzaufgabe:**

Schreiben Sie ein Assemblerprogramm ( Funktion: `int multi(unsigned byte, unsigned byte)` ), welches das Produkt zweier 8 bit vorzeichenloser Zahlen berechnet und in R0 zurück liefert. Testen und dokumentieren Sie Ihr Programm.

Der erforderliche Praktikumsbericht dient zu Ihrer Nachbereitung des Praktikums. Er beinhaltet die Formulierung der Lösungsidee, die Angabe der Größe der Programme in Bytes sowie den zeilenweisen kommentierten Quelltext. Haben Sie die Praktikumsberichte, für eine evtl. Kontrolle durch die Betreuer, dabei. Die Erstellung eines Berichts für jede Gruppe ist erlaubt.

## Zu Aufgabe 1:

// Name: Matrikelnummer:  
// Name: Matrikelnummer:  
// Datum:

```
.file "aufgabe1.S"
.text @ legt eine Textsection fuer PrgrammCode + Konstanten an
.align 2 @ sorgt dafuer, dass nachfolgende Anweisungen auf einer durch 4 teilbaren
Adresse liegen
        @ unteren 2 Bit sind 0
.global main @ nimmt das Symbol main in die globale Sysmboltabelle auf
.type main,function
main:
MOV R0, R1, LSR #2 @ ...
ADD R0, R1, R1, LSL #3 @ ...

bx lr @ Ruecksprung zum aufrufenden Programm
.Lfe1:
.size main,.Lfe1-main@ Programmgroesse berechnen

// End of File
```

## Zu Aufgabe 2:

```
// Name:           Matrikelnummer:  
// Name:           Matrikelnummer:  
// Datum:
```

```
    .file    "aufgabe2.S"  
    .text    @ legt eine Textsection fuer PrgrammCode + Konstanten an  
    .align  2    @ sorgt dafuer, dass nachfolgende Anweisungen auf einer durch 4 teilbaren  
Adresse liegen  
                @ unteren 2 Bit sind 0  
    .global  main    @ nimmt das Symbol main in die globale Sysmboltabelle auf  
    .type   main,function  
main:  
    MOV    r0, #0  
    ADDS   r0, r0, r0    @ ...  
  
    bx     lr  
.Lfe1:  
    .size  main,.Lfe1-main  
  
// End of File
```

### Zu Aufgabe 3:

// Name: Matrikelnummer:  
// Name: Matrikelnummer:  
// Datum:

```
.file "aufgabe3.S"
.text @ legt eine Textsection fuer PrgrammCode + Konstanten an
.align 2 @ sorgt dafuer, dass nachfolgende Anweisungen auf einer durch 4 teilbaren
@ Adresse liegen
@ unteren 2 Bit sind 0
.global main @ nimmt das Symbol main in die globale Sysmboltabelle auf
.type main,function
main:
push {r4, r5, r6, r7, r9, lr}

ldr R0, = 0xaabbccdd
ldr R1, = 0xffbbffbb
ldr R2, = 0xffffffff
ldr r3, = 0x123456 @ z.B. Matrikelnummer
ldr r4, = 0x3
ldr r5, = 0x2
ldr r6, = 0x7ffffff
ldr r7, = 0x80000000
@ R9 (hexadez.) - N, Z, C, V
ANDS R9, R0, R3 @ - Wie werden die Flags N, Z, C, V gesetzt? _ _ _ _
EOR R9, R3, R3 @ - Gilt das Ergebnis für jeden Wert in R3? ja / nein
SUBS R9, R7, #3 @ - Wie werden die Flags N, Z, C, V gesetzt? _ _ _ _
RSBS R9, R5, #3 @ - Wie werden die Flags N, Z, C, V gesetzt? _ _ _ _
ADDS R9, R4, #12 @ - Wie werden die Flags N, Z, C, V gesetzt? _ _ _ _
ADDS R9, R6, R4 @ - Wie werden die Flags N, Z, C, V gesetzt? _ _ _ _
TST R4, #1 @ - Wie werden die Flags N, Z, C, V gesetzt? _ _ _ _
TEQ R4, R4 @ - Wie werden die Flags N, Z, C, V gesetzt? _ _ _ _
CMP R5, R4 @ - Wie werden die Flags N, Z, C, V gesetzt? _ _ _ _
CMN R2, R5 @ - Wie werden die Flags N, Z, C, V gesetzt? _ _ _ _
ORR R9, R0, R3 @
MOV R9, #126 @
BIC R9, R0, R1 @
BIC R9, R2, #15 @
MVN R9, R1 @

pop {r4, r5, r6, r7, r9, pc}
.Lfe1:
.size main,.Lfe1-main

// End of File
```

## zu Aufgabe 4:

// Name:           Matrikelnummer:

// Name:           Matrikelnummer:

// Datum:

```

        .file      "aufgabe4.S"
        .text      @ legt eine Textsection fuer PrgrammCode + Konstanten an
        .align    2      @ sorgt dafuer, dass nachfolgende Anweisungen auf einer durch 4 teilbaren Adresse
liegen
                @ unteren 2 Bit sind 0
        .global   main    @ nimmt das Symbol main in die globale Symboltabelle auf
        .type    main,function
main:
        push    {r4, r5, lr}
        mov     r0, #1
        mov     r1, #-1
        mov     r2, #15
        mov     r3, #0x80000000
//..

        pop     {r4, r5, pc}
.Lfe1:
        .size   main,.Lfe1-main

// End of File
```

### zu Zusatzaufgabe:

```
//  
// Loesung zu int multi(unsigned byte, unsigned byte)  
//  
//Name:  
//Datum:  
  
    .file    "multi.S"  
    .text  
    .align  2  
    .global multi  
    .type   multi,function  
multi:  
// In den Registern R0 und R1 werden Multiplikant und Multiplikator uebergeben  
  
// In Register R0 wird das Produkt zurück gegeben  
    bx     lr  
.Lfe1:  
    .size  multi,.Lfe1-multi  
  
// Zum Testen der in Assembler zu schreibenden Funktionen "int multi(unsigned byte, unsigned byte)"  
// von Manfred Pester  
// vom 12.02.2013  
//  
int multi(unsigned byte, unsigned byte);  
  
int main (void)  
{  
    unsigned byte  ubyte_multiplikant = 5;  
    unsigned byte  ubyte_multiplikator = 51;  
    unsigned int   uint_ergebnis = 0;  
  
        ergebnis = multi(multiplikant, multiplikator);  
  
    return 0;  
}
```

```
# makefile für Rechnerarchitekturpraktikum Termin 3 WS2013  
# von: Manfred Pester  
# vom: 12.02.2013
```

```
# Variable fuer den zu nutzenden Compiler  
GCC = arm-elf-eb63-gcc
```

```
all: aufgabe1 aufgabe2 aufgabe3 aufgabe4 multi zusatzaufgabe
```

```
aufgabe1: aufgabe1.S  
    $(GCC) -g aufgabe1.S -o aufgabe1.elf
```

```
aufgabe2: aufgabe2.S  
    $(GCC) -g aufgabe2.S -o aufgabe2.elf
```

```
aufgabe3: aufgabe3.S  
    $(GCC) -g aufgabe3.S -o aufgabe2.elf
```

```
aufgabe4: aufgabe4.S  
    $(GCC) -g aufgabe4.S -o aufgabe4.elf
```

```
multi: multi.S  
    $(GCC) -g -c multi.S
```

```
zusatzaufgabe: zusatzaufgabe.c multi.S  
    $(GCC) -S zusatzaufgabe.c  
    $(GCC) -g zusatzaufgabe.c multi.S -o zusatzaufgabe.elf
```

```
clean:  
    rm *.o  
    rm *.elf
```