



h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi
FACHBEREICH INFORMATIK

PRAKTIKUM
RECHNERARCHITEKTUR
WS2023
Termin 3
ARM: Arithmetische und logische Operationen

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

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

Ziel der folgenden Aufgaben:

Verständnis für arithmetische und logische Operationen und die Flags im Statusregister. Weiteres Ziel ist die selbstständige 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	$Rd = Op1 \text{ AND } Op2$
EOR	$Rd = Op1 \text{ EOR } Op2$
SUB	$Rd = Op1 - Op2$
RSB	$Rd = Op2 - Op1$
ADD	$Rd = Op1 + Op2$
ADC	$Rd = Op1 + Op2 + \text{Carry}$
SBC	$Rd = Op1 - Op2 - \text{Carry}$
RSC	$Rd = Op2 - Op1 - \text{Carry}$
TST	setzt Condition Codes bzgl. $Op1 \text{ AND } Op2$
TEQ	setzt Condition Codes bzgl. $Op1 \text{ EOR } Op2$
CMP	setzt Condition Codes bzgl. $Op1 - Op2$
CMN	setzt Condition Codes bzgl. $Op1 + Op2$
ORR	$Rd = Op1 \text{ ORR } Op2$
MOV	$Rd = Op2$
BIC	$Rd = Op1 \text{ AND NOT } Op2$
MVN	$Rd = \text{NOT } Op2$ (Einerkomplement)

Bereiten Sie die folgenden Aufgaben so vor, dass Sie die Ergebnisse und Programme inklusive Dokumentation zum Laborterminal präsentieren können.

Zur Erreichung des Testats müssen Sie die fertigen Programme vorführen und erklären können.

Aufgabe 1:

Was leisten die folgenden beiden Befehle?

LSL R0, R1, #3

_____ Was passiert mit negativen Zahlen?

ADD R0, R1, R1, ASR #1

_____ Diskutieren Sie die Wertebereiche

Aufgabe 2:

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

Vermeiden Sie hierbei - sofern möglich - das gleichzeitige Setzen von mehreren Flags und versuchen Sie eine Lösung zu erarbeiten, welche nur das entsprechende Flag setzt!

Sie können beliebige Werte in beliebigen Registern nutzen.

Beschriften und Befüllen Sie die von Ihnen gewählten Register neben der jeweiligen Zeile, damit klar wird, mit welchen Werten Sie rechnen.

Sie können gerne auch #Immediate-Operanden nutzen, anstelle eines zweiten Registers!

Beispiel (finden Sie für das N-Flag auch noch eine **eigene** Lösung!):

Registers		Flag
R ₁ = 0x00000001	R ₂ = 0x80000000	N: ADDS R0, R1, R2

Registers		Flag
R_ =	R_ =	N: _____
R_ =	R_ =	Z: _____
R_ =	R_ =	C: _____
R_ =	R_ =	V: _____

(fürs V-Flag wären auch 2 Befehle erlaubt.
 Schauen Sie sich zuvor Aufgabe 3 an.)

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			

Aufgabe 4:

Überprüfen Sie mit den gegebenen Programmen „aufgabe1.S“ bis „aufgabe3.S“ Ihre Lösungen der Aufgaben 1 bis 3.

Wechseln Sie in das Verzeichnis raWS2023/Termin3/. Starten Sie den snavigator und legen ein neues Projekt an. Beachten Sie dazu die Dokumentation zur Entwicklungsumgebung.

Sofern die Testprogramme andere Ergebnisse liefern: Analysieren Sie warum dies der Fall ist. Finden Sie die Fehler und berichtigen diese.

Aufgabe 5:

Es sind in den Registern R0 bis R3 „signed integer“-Werte gegeben. Überlegen Sie sich mindestens vier mathematische universell einsetzbare verschiedene Arten, wie Sie eine Vorzeichenumkehr bei Erhaltung des Betrags erreichen können -- zum Beispiel indem Sie die Schritte der 2er-Komplementbildung nachbilden. Programmieren, testen und dokumentieren Sie Ihre Verfahren und Erkenntnisse.

Zusatzaufgabe 1:

Schreiben Sie ein ARM-Assembler-Programm, welches den Inhalt von zwei beliebigen Registern tauscht, ohne zusätzliche (neben den zwei zu tauschenden) Register oder Speicherstellen zu verwenden.

Versuchen Sie so wenige Codezeilen wie möglich zu erreichen.

Zusatzaufgabe 2:

Schreiben Sie ein ARM-Assembler-Programm, welches aus jedem Bit des im Register 0 gegebenen Wert das ASCII-Zeichen „0“ oder „1“ codiert und diese in einem 32 Zeichen großen String ablegt.

Beispiel: R0 = 0x12345678 → String .byte 0x00, 0x00, 0x00, 0x31, 0x30, 0x30, 0x31, 0x30, 0x30, 0x30, 0x30, 0x31, 0x31, ..., 0x31, 0x30, 0x30, 0x30

Überprüfen Sie Ihr Programm darauf, ob Sie es mit noch weniger Code-Zeilen umsetzen können.

Der erforderliche Praktikumsbericht dient zu Ihrer Nachbereitung des Praktikums. 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:
    LSL    R0, R1, #3      @ ...
    ADD    R0, R1, R1, ASR #1 @ ...

    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 ProgrammCode + 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:
    MOV r1, #1
    MOV r2, #0x80000000

    ADDS r0, r1, r2    @ ...
// ...

    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 ProgrammCode + 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 5:

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

```
.file    "aufgabe5.S"
.text    @ legt eine Textsection fuer ProgrammCode + 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, r6, r7, lr}
    mov     r4, #1
    mov     r5, #-1
    mov     r6, #15
    mov     r7, #0x80000000
//..

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

// End of File
```

zu Zusatzaufgabe 1:

```
// Name:          Matrikelnummer:
// Name:          Matrikelnummer:
// Datum:
    .file    "zusatzaufgabe1.S"
    .text    @ legt eine Textsection fuer ProgrammCode + 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 zusatzaufgabe1 in die globale
Symboltabelle auf
    .type    main, function
```

main:

```
    bx    lr
.Lfe1:
    .size  main, .Lfe1-main
```

// End of File

zu Zusatzaufgabe 2:

```
// Name:          Matrikelnummer:
// Name:          Matrikelnummer:
// Datum:
    .file    "zusatzaufgabe2.S"
    .text    @ legt eine Textsection fuer ProgrammCode + 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 zusatzaufgabe1 in die globale
Symboltabelle auf
    .type    main, function

main:
    ldr     r0, = 0x87654321

    bx     lr
.Lfe1:
    .size   main, .Lfe1-main

// Speicherplatz fuer einen 0-terminierten String mit 32 Zeichen
    .comm   Ziffern_String, 33

// End of File
```

```
# Makefile für Rechnerarchitekturpraktikum Termin 3 WS2023
# von: Manfred Pester
# vom: 26.09.2022
```

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

```
all: aufgabe1 aufgabe2 aufgabe3 aufgabe5
```

```
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 aufgabe3.elf
```

```
aufgabe5: aufgabe5.S
    $(GCC) -g aufgabe5.S -o aufgabe5.elf
```

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