PRAKTIKUM

RECHNERARCHITEKTUR

WS2020

Termin 3

ARM: Arithmetische und logische Operationen

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 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)

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

Aufgabe 1:

Was leisten die folgenden beiden Befehle?

LSR R0, R1, #2 _____

ADD R0, R1, R1, LSL #3 _____

Aufgabe 2:

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

Im Register R0 steht 0x1 und im Register R1 steht 0x80000000.

Beispiel: ADDS R2, R0, R1 @setzt z.B. nur das Vorzeichen-Flag

_____

_____

_____

_____

_____

Aufgabe 3:

Füllen Sie die untenstehende Tabelle aus.

Die Register haben folgende Werte:

R0 = 0xAABBCCDD

R1 = 0xFFBBFFBB

R2 = 0xFFFFFFFE

R3 = = 0x123456

R4 = 0x3

R5 = 0x2

R6 = 0x7fffffff

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.
Sofern die Testprogramme andere Ergebnisse liefern: Analysieren Sie, warum dies der Fall ist.

Aufgabe 5:

Es sind in den Registern R0 bis R3 Werte gegeben, deren Vorzeichen Sie auf verschiedene Arten (z.B. 2K-Wandlung) umkehren sollen. Überlegen Sie sich mindestens drei weitere universell einsetzbare Verfahren, programmieren, testen und dokumentieren Sie Ihre Verfahren und Erkenntnisse.

Aufgabe 6:

Schreiben Sie ein ARM-Assembler-Programm, welches Byte1 und Byte3, sowie Byte2 und Byte4 in einem Register vertauscht. Beispiel: 0x1234ABCD -> 0xCDAB3412.

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

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 benötigen.

Zu Aufgabe 1:

```
        .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
main:   .type   main,function

        LSR R0, R1, LSR #2      @ ...
        ADD R0, R1, R1, LSL #3 @ ...

.Lfe1:  bx      lr      @ Ruecksprung zum aufrufenden Programm

        .size   main,.Lfe1-main@ Programmgroesse berechnen

// End of File
```

****

Zu Aufgabe 2:

```
        .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 Symboltabelle auf
main:   .type   main,function

        MOV     r0, #1
        MOV     r1, #0x80000000

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


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

// End of File
```

****

Zu Aufgabe 3:

```
        .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, = 0xfffffffe
        ldr     r3, = 0x123456
        ldr     r4, = 0x3
        ldr     r5, = 0x2
        ldr     r6, = 0x7fffffff
        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     @
.Lfe1:  MVN     R9, R1          @
        pop     {r4, r5, r6,r7, r9, pc}
        .size   main,.Lfe1-main

// End of File
```

****

zu Aufgabe 5:

```
        .file   "aufgabe5.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 Aufgabe 6:

```
        .file   "aufgabe6.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:   ldr r1, =0x1234ABCD
//..

        bx      lr
.Lfe:   .size   main,.Lfe1-main
// End of File
```

****

```
# makefile für Rechnerarchitekturpraktikum Termin 3 WS2020

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

all: aufgabe1 aufgabe2 aufgabe3 aufgabe5 aufgabe6

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

    Aufgabe6: aufgabe6.S
        $(GCC) -g aufgabe6.S -o aufgabe6.elf

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