h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi
FACHBEREICH INFORMATIK

RECHNERARCHITEKTUR

WS16/17

Termin3

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 das Verständnis der arithmetischen und logischen Operationen, das Rechnen mit Prozessorregistern 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)
LSL	Rd = Op1 << Op2
LSR	Rd = Op1 >> Op2 (logisches Shift für unsigned)
ASR	Rd = Op1 >> Op2 (arithmetisches Shift für signed)
ROR	Rd = Op1 rot Op2 (Rotation nach rechts)

Aufgabe 1:

Füllen Sie die unten stehende Tabelle aus indem sie das Programm Vorbereitung.S im Debugger ausführen

Überprüfen sie die Operationen, indem sie einige Werte mit dem Taschenrechner nachrechnen. Welche der Werte können sie im Kopf ausrechnen?

Die Register haben folgende Werte:

MOV R0, =0xAABBCCDD

R1 = 0xFFBBFFBB

R2 = 0xFFFFFFFE

R3 = 0x55563776

R4 = 0x3

R5 = 0x2

R6 = 0x7fffffff

R7 = 0x80000000

Instruktion	R9 (hexadez.)	Zusatzfrage	Antwort
ANDS R9, R0, R3			
EORS R9, R3, R3		Gilt das Ergebnis für jeden Wert in R3?	Ja/Nein
SUBS R9, R7, #3			
RSBS R9, R5, #3			
ADDS R9, R4, #12			
ADDS R9, R6, R4			
CMP R5, R4		Wie werden die Flags N, Z, C, V gesetzt?	_, _, _, _
CMN R2, R5		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?	_, _, _, _
ORR R9, R0, R3			
MOV R9, #126			
BIC R9, R0, R1			
BIC R9, R2, #15			
MVN R9, R1			
LSL R9, R3, #16			
LSR R9, R3, #8			
ASR R9, R7, #16			
ROR R9, R3, #16			

Arbeitsverzeichnis:

Holen sie sich von der Webseite

https://www.fbi.h-da.de/organisation/personen/raffius-gerhard/rechnerarchitektur.html

den Zip Ordner mit den Augaben. Entpacken sie den Ordner und holen sie das Verzeichnis termin3. Kopieren sie es sich in ein Arbeitsverzeichnis (keine Leerzeichen im Pfadnamen) und führen das Kommando

make Vorbereitung.elf

in einer Konsole aus. Mit dem Aufruf

arm-elf-insight Vorbereitung.elf

starten sie den Debugger und untersuchen die Ergebnisse

Aufgabe 2:

Bei einer BCD codierten Zahl werden in 4 Bits (1 Nibble) jeweils eine dezimale Ziffer gespeichert. Der Vorteil ist, dass die Wandlung von BCD Zahlen in das Dezimalsystem und umgekehrt sehr einfach ist. Nachteilig ist, dass die normalen Rechenfunktionen wie Addition und Subtraktion deutlich schwieriger sind. Die BCD Kodierung war früher weit verbreitet und die ersten Computer und Taschenrechner nutzten sie zur Zahlendarstellung.

Eine 32 Bit Integerzahl besitzt 8 Nibbles.

$$\text{Intzahl} = \sum_{i=1}^{8} nib_i * 10^{i-1}$$

Die Wandlung von BCD zu Integer erfolgt indem man den höchstwertigen Koeffizienten mit 10 multipliziert und anschließend den nächstniedrigeren Nibble-Koeffizienten addiert, vom höchstwertigen Nibble ausgehend, bis alle Nibble verarbeitet sind. Dieser Algorithmus wird das Hornerschema genannt.

$S = (...((a_8*10 + a_7) * 10 + a_6) * 10 ...) + a_1$

Programmtechnisch ist es einfacher die Summe zuerst auf Null zu setzen. Dann multipliziert man in einer Schleife zuerst die Summe mit 10 und addiert anschließend den jeweiligen Koeffizienten hinzu

$S = 0$

for (i = 8..1) $S = S * 10 + a_i$

Der Algorithmus zur Polynomberechnung wird so häufig benutzt, dass man für die benötigte Rechenoperation einen eigenen Befehl eingeführt hat. Der MLA Befehl führt eine Multiplikation mit anschließender Addition aus

```
mla Rd, Rn, Rm, Ra
```

Der Befehl hat als Randbedingung, dass alle Werte in Registern stehen müssen, und dass Rd nicht gleich Rn sein darf. Die zweite Bedingug ist nicht tragisch, da die Multiplikation zum Glück kommutativ ist, so dass man bei Bedarf Rn und Rm vertauschen kann.

Schreiben sie das Assemblerprogramm bcd2int(int) das eine BCD codierte Zahl übergeben bekommt und daraus eine normale Integerzahl bildet.

Überlegen sie sich, wie sie den MLA Befehl durch zwei Additionsbefehle mit Shift des 2. Operanden ersetzen könnten. (Multiplikation mit 10 ist Multiplikation mit 5=(4+1) gefolgt von Multiplikation mit 2)

Studieren sie vorher die Funktionen bcd2str(char*, int) und str2bcd(int). Führen sie das Testprogramm aus und untersuchen die Funktionsweise der beiden Programme im Debugger im Einzelschrittmodus. Wenn sie beide Programme verstanden haben, überlegen sie, wie sie die gewonnenen Erkenntnisse Nutzen können um das Programm bcd2int zu schreiben.

Beachten sie, dass das C Programm den ersten Parameter an das Unterprogramm in R0 übergibt. Im Unterprogramm dürfen sie die Register R0-R3 und IP benutzen und deren Werte verändern. Die Rückgabe des Ergebnisses erfolgt in Register R0.

Leider enthält das Programm bcd2str noch einen versteckten Fehler. Finden sie den Fehler und machen sie es in ihrem Programm bcd2int besser. Das Testprogramm zeigt ihnen den Fehlerfall an.

Das zugehörige Makefile unterstützt sie bei der Übersetzung der Programme.

make test.elf

arm-elf-insight test.elf

Das File zum Testen der Wandlung

```
// Zum Testen der in Assembler zu schreibenden Funktionen "int bcd2int(int)"
// von Gerhard Raffius
// vom 28.3.2016
//

#include <stdio.h>

unsigned char* testvec[] = {"11111111", "11110000", "1234", "0000", "ffff", "1984"};
unsigned char buf[20];

int bcd2int(int);
int str2bcd(unsigned char*);
void bcd2str(unsigned char*, int);

int main (void)
{
	int i, bcd, ergebnis;
	for( i=0; i < sizeof(testvec)/sizeof(char*); i++ ) {
		bcd = str2bcd(testvec[i]);
		ergebnis = bcd2int( bcd );
		bcd2str(buf, bcd);
		printf("Test:  %s == %d\n", buf, ergebnis);
	}
	return 0;
}
```

Die kommentierte Vorlage zum Programm bcd2int

```
@ Name:        Matrikelnummer:
@ Name:        Matrikelnummer:
@ Datum:

        .file  "bcd2int.S"    @ diese Zeile braucht der Debugger
        .text                 @ legt eine Textsection fuer ProgrammCode + Konstanten an
        .align 2              @ sorgt dafuer, dass nachfolgende Anweisungen auf einer durch
                                      @ 4 teilbaren Adresse liegen

        .global       bcd2int@ nimmt das Symbol bcd2int in die globale Sysmboltabelle auf

        .type  bcd2int,function     @ braucht der Linker

bcd2int:
        @ In Register R0 wird BCD Zahl uebergeben
        @ In Register R0 soll aber auch die int-Zahl zurueck gegeben werden,
        @ also am besten den Parameter in ein anderes Register kopieren
        @ die Register R0-R3 und IP stehen zur freien Benutzung zur Verfügung



loop:  @ dies ist ein Label. Wir können Labels als Sprungziele benutzen



        bx     lr     @ dies ist der Rücksprung, da der gerettete Programcounter hoffentlich
                      @ noch in LR steht

.Lfe1:
        .size bcd2int,.Lfe1-bcd2int

@ End of File
.end   @ dies muss in der letzten Zeile stehen
```

Die Hilfsprogramme bcd2str und str2bcd

```
str2bcd:
       mov r1, r0           @ 1. Parameter steht in r0,
                            @ da am Ende Ergebnis in r0 stehen soll,
                            @ wird Parameter nach r1 verschoben
       mov r0, #0           @ r0 mit Anfangswert laden

loop:
       ldrb r2, [r1], #1            @ ersten Wert nach r2 holen
                                    @ und Zeiger danach inkrementieren
       cmp r2, #0                   @ Abfrage, ob End of String erreicht ist
       beq end_loop                 @ wenn ja, Schleife verlassen
       cmp r2, #'9'                 @ Abfrage ob Ziffer
       suble r2, r2, #'0'           @ Subtraktion der Ascii 0 wenn Ziffer
       cmp r2, #'a'                 @ Abfrage ob Buchstabe
       subge r2, r2, #('a'-10)      @ Korrektur der Hexadezimalzeichen a-f
       add r0, r2, r0, lsl #4       @ bisheriges Ergebnis shiften und
                                    @ neue BCD Ziffer addieren
       b loop
end_loop:            @ Ergebnis steht jetzt in r0 (APCS Ergebniswert)

                     @ In Register R0 wird die int-Zahl zurueck gegeben
       bx     lr

bcd2str:
       @ In Register R0 wird Zeiger auf Buffer uebergeben (APCS erster Parameter)
       @ in Register R1 wird die BCD Zahl übergeben (APCS zweiter Parameter)

loop1:
       mov r1, r1, ror #28          @ Oberstes Nibbel in richtige Position bringen
       and r2, r1, #0xf             @ Nibbel isolieren
       cmp r2, #10                  @ auf Ziffer überprüfen
       addlt r2, #'0'               @ Ascii Null addieren
       addge r2, #('a' -10)         @ oder nach a-f umwandeln
       strb r2,[r0], #1             @ Ergebnis in String speichern und
                                    @ Zeiger inkrementieren
       bics r1, r1, #0xf            @ verarbeiteten Nibbel löschen und auf Null überprüfen
       bne loop1                    @ zurück, wenn noch etwas zu verabeiten ist
       mov r2, #0                   @ Endemarkierung nach r2
       strb r2,[r0]                 @ String mit 0 abschliessen

                                    @ keine Rückgabe, da void Funktion
       bx     lr
```