Rechnerorganisation

Sommersemester 2023

Prof. Stefan Roth, Ph.D.

TECHNISCHE UNIVERSITÄT DARMSTADT

4. Aufgabenblatt mit Lösungsvorschlag

15.5.2023
Konzepte der maschinennahen Programmierung, Arrays

Aufgabe 1: Theoriefragen

(a) In welche Register wird die Rücksprungadresse bei Ausführung des Befehls bl gespeichert?

(b) Was ist der Unterschied zwischen einer Funktion und einer Prozedur?

(c) Was ist der Unterschied zwischen Makrotechnik und Unterprogrammtechnik?

Lösungsvorschlag:

(a) Im Register r14 (`lr`)

(b) Eine Funktion hat einen Rückgabewert, eine Prozedur hat keinen Rückgabewert.

(c) Bei der Makrotechnik wird das Teilprogramm an den Stellen, wo es gebraucht wird, einkopiert. Bei der Unterprogrammtechnik hingegen ist das Teilprogramm nur einmal im vorhanden und wird über Sprünge (z.B. mit `bl`) aufgerufen.

Aufgabe 2: Array Sortierung

1. Implementieren Sie in C einen Algorithmus der überprüft, ob die Elemente eines Arrays aufsteigend sortiert sind. Falls es sortiert ist, soll am Ende eine 1 ausgegeben werden, sonst eine 0. Testen Sie Ihr Programm mit unterschiedlichen Arrays. Kommentieren Sie Ihren Code.

Lösungsvorschlag:

```c
#include <stdio.h>
#include <stdbool.h>

int main()
{
    int feld [5] = {1,2,3,3,5};  /* das zu ueberpruefende Array */
    int laenge = sizeof(feld)/sizeof(feld[0]); /* berechnet die Laenge von feld */

    bool is_sorted = 1;                  /* der Rueckgabewert auf 1 gesetzt */

    for (int i = 0; i < laenge-1; i++) /* Iteration ueber die Elemente von feld */
    {
        if (feld[i] > feld[i+1]) /* Vergleich ob zwei benachbarte Elemente sortiert sind */
        {
            is_sorted = 0; /* wenn Elemente nicht sortiert is_sorted = 0 */
        }
    }
    printf("%d \n", is_sorted);
    return 0;
}
```

2. Implementieren Sie den obigen Algorithmus in ARM-Assembler. Falls das Feld sortiert ist, soll am Ende in r0 eine 1 gespeichert werden, sonst eine 0. Testen Sie Ihr Programm mit unterschiedlichen Arrays. Kommentieren Sie Ihren Code.

Lösungsvorschlag:

```
/* -- Array.s */
/* Kommentar  */

.data /* Daten Bereich */
feld: .word 1,2,3,4,5 /* Feld */
laenge: .word 5 /* laenge, Wert 5 */

.global main /* Definition Einsprungpunkt Hauptprogramm */

/* Der Algorithums vergleicht jeweils zwei benachbarte Werte
und gibt 0 zurueck, wenn der erste Wert groesser ist als der
zweite. Sonst werden die naechsten beiden Werte verglichen,
solange bis das Array komplett durchlaufen wurde. */

main:           /* Hauptprogramm */
    ldr r0,adr_feld /* laedt Adresse von feld in r0 */
    ldr r1,adr_laenge /* laedt Adresse von laenge in r1 */
    ldr r1,[r1] /* laedt Wert von laenge in r1 */
    lsl r1, r1, #2 /* laenge wird mit 4 multipliziert, da auf
    Woerter in einem byte-adressierten Speicher zugegriffen
    wird */

    mov r2,#0 /* laedt den ersten offset in r2 */
    mov r3,#4 /* laedt den zweiten offset in r3 */

    WHILE: /* Start der Schleife */
    cmp r3, r1 /* ueberprueft, ob das array komplett durchlaufen wurde */
    bge sorted /* springt zu sorted wenn der zweite offset groesser/gleich laenge ist */

    ldr r4,[r0, r2] /* laedt den ersten Wert in r4 */
    ldr r5,[r0, r3] /* laedt den zweiten Wert in r5 */
    cmp r4, r5 /* vergleicht die beiden Werte miteinander */
    bgt unsorted /* springt zu unsorted, wenn wert1 > wert2 */

    add r2, r2, #4 /* berechnet die neuen Offsets */
    add r3, r3, #4
    b WHILE /* springt zum Anfang der Schleife */

    sorted:
    mov r0, #1 /* laedt 1 in r1 */
    b end
    unsorted:
    mov r0, #0 /* laedt 0 in r1 */

    end:
    bx lr       /* Springt zurueck zum aufrufenden Programm */

adr_feld: .word feld /* Adresse von feld */
adr_laenge: .word laenge /* Adresse von laenge */
```

Aufgabe 3: Bitmanipulation

Zum Schutz vor illegalen Kopien verlangt ein Programm bei der Installation eine achtstellige (hexadezimale) Seriennummer. Um die Gültigkeit einer solchen Seriennummer zu prüfen, werden die Bits der Stellen 7 bis 9 und 23 bis 25 jeweils als eine Zahl interpretiert. Wenn die Summe dieser beiden Zahlen 12 beträgt, ist die Seriennummer gültig. Hierbei betrachten wir das niederwertigste Bit als Bit mit Index 0.

1. Gegeben sei die Seriennummer $42C27F91_{16}$. Zeigen Sie, dass dies eine gültige Seriennummer ist.

Lösungsvorschlag:

Aus der Binärdarstellung `0100 0010 1100 0010 0111 1111 1001 0001` geht hervor, dass an der 7 bis 9 Stelle der Binärwert $a = 111_2$ und an der 23. bis 25. Stelle der Binärwert $b = 101_2$ steht. Addition dieser beiden Werte ergibt 1100_2, was dem Wert 12 entspricht. Damit ist die angegegebene Seriennummer gültig.

2. Wie viele gültige Seriennummern gibt es?

Lösungsvorschlag:

Eine gültige Seriennummer hat folgende Form: `XXXX XXaa aXXX XXXX XXXX XXbb bXXX XXXX`. Die Bit-Werte der mit `X` markierten Stellen sind egal, und die Werte `aaa` und `bbb` müssen so gewählt sein, dass `aaa` + `bbb` = 12. Durch Probieren findet man drei Möglichkeiten, um die Zahl 12 als Summe zweier 3-Bit-Zahlen darzustellen: $6+6$, $5+7$ und $7+5$. Für die 6 Bits `aaa` und `bbb` gibt es also 3 mögliche Kombinationen, während die übrigen $32-6=26$ Bits beliebige Werte annehmen können. Folglich gibt es $3 \cdot 2^{26}$ gültige Seriennummern.

3. Erzeugen Sie eine weitere gültige und eine ungültige Seriennummer. Zeigen Sie deren Gültigkeit bzw. Ungültigkeit.

Lösungsvorschlag:

Eine gültige Seriennummer ist z. B. `1111 0111 0101 0000 0111 1011 0101 1101`; eine ungültige ist z.B. `0100 0000 1100 0110 1000 1001 0001 0001`.

4. Schreiben Sie in ARM-Assembler ein Programm, das die Gültigkeit einer Seriennummer prüft. Nehmen Sie hierzu an, dass die Seriennummer zu Beginn im Register r1 gespeichert ist. Ist diese Seriennummer gültig, soll das Register r0 am Endes des Codes den Wert 1 enthalten, andernfalls den Wert 0.

   Hinweis: Zur Implementierung dieses Programms ist es sinnvoll, sich in einem sogenannten Reference Guide [1] Maschinenbefehle zur Bitmanipulation anzuschauen. Die Schiebebefehle (lsl, lsr) haben Sie bereits kennengelernt. Ein weiterer Befehl ist z. B. and. Sie finden die Befehlsübersicht der 32 Bit Befehle ab Seite 7 ff.

Lösungsvorschlag:

```
/* -- sn.s */
/* Kommentar  */
.data /* Daten Bereich */
sn: .word  0x42C27F91
bit1: .word 0x00000380
bit2: .word 0x03800000

.global main /* Definition Einsprungpunkt Hauptprogramm */

main:           /* Hauptprogramm */
    ldr r0, adr_sn /* r0 als Basisadressregister */
    ldr r1, [r0] /* lade Seriennummer in Register r1 */
    ldr r0, adr_bit1 /* lade Adresse von Bitmaske1 */
    ldr r2, [r0] /* lade Bitmaske1 in r2 */
    and r1,r1,r2 /* Maskiere die Bits aus */
    lsr r1,r1,#7 /* erstes Teilergebnis */
```

[1] Arm Instruction Set Reference Guide - https://moodle.tu-darmstadt.de/pluginfile.php/1809291/mod_folder/content/0/Material/arm_instruction_set_reference_guide.pdf

```
    ldr r0, adr_sn /* r0 als Basisadressregister */
    ldr r3,[r0] /* lade Seriennummer in Register r3 */
    ldr r0, adr_bit2 /* lade Adresse von Bitmaske2 */
    ldr r2, [r0] /* lade Bitmaske2 in r2 */
    and r3,r3,r2 /* Maskiere die Bits aus */
    lsr r3,r3,#23 /* zweites Teilergebnis */

    add r4,r1,r3 /* Addition der Teilergebnisse */

    mov r0,#0
    cmp r4,#12 /* pruefe, ob Ergebnis 12 */
    bne ende /* not equal */
    mov r0,#1 /* gebe die 1 zurueck, wenn SN gueltig */
ende:
    bx lr       /* springe zurueck zum aufrufenden Programm */

adr_sn: .word sn /* Adresse von sn */
adr_bit1: .word bit1 /* Adresse von bit1 */
adr_bit2: .word bit2 /* Adresse von bit2 */
```