
Test

Hardwarenahe Softwareentwicklung

6. Mai 2022

Name: _____

Klausur-ID: 10

Aufgabe	1	2	3	4	Σ
erreichte Punkte					
von maximal	3	3	8	6	20

Prüfungsnote (50%)	
Erfahrungsnote (50%)	

Notenschnitt	
Modulnote	

Hinweise:

- ▶ Schreiben Sie bitte **Ihren Namen auf dieses Titelblatt**.
- ▶ Dieser Test umfasst 4 Aufgaben. Kontrollieren Sie bitte **unmittelbar nach Beginn des Tests**, ob alle Aufgaben vorhanden sind.
- ▶ Der Test dauert **60 Minuten**.
- ▶ Lösen Sie bitte **jede Aufgabe auf dem dafür vorgesehenen Blatt**. Wenn Sie weitere Blätter benutzen, versehen Sie das Blatt mit **Klausur-ID** und **Aufgabennummer**.
- ▶ Programmieraufgaben können Sie optional elektronisch abgeben. In diesem Falle ist im Datei-Header zwingend Ihr Name anzufügen.
- ▶ Dieser Test beinhaltet auch Multiple-Choice-Fragen. Die einzelnen Multiple-Choice-Fragen sind voneinander unabhängig. Pro Frage können auch mehrere Antworten möglich sein. Bewertungsschema:
 - Eine vollständig richtig beantwortete Frage gibt 0.5 Punkte.
 - Eine teilweise korrekt beantwortete Frage gibt 0.25 Punkte, wobei falsche Teilantworten negativ gewichtet werden. Eine Frage kann aber total keine negative Anzahl Punkte geben.
 - Eine nicht beantwortete Frage gibt 0 Punkte.
- ▶ Zuzulässige Hilfsmittel:

- Skripte und Musterlösungen, ergänzt durch eigene Notizen und eigene Übungslösungen.
- Befehlssatz ARM Cortex-Mx
- Laptop, Taschenrechner

Viel Erfolg!

Aufgabe 1: Multiple Choice Hardware (je 0.5 Punkte, total 3 Punkte)

-
- a) Welche Aussagen zur "Von Neumann Architektur" sind korrekt?
- ☐ Daten und Programmcode liegen in unterschiedlichen Adressräumen.
 - ☐ Es gibt einen Bus für die Daten und einen für Programminstruktionen.
 - ☐ Die CPU liest Daten und Programminstruktionen sequenziell ein.
 - ☐ Es gibt genau einen Datenbus.
-
- b) Welche Aussagen zu Cortex-M7 sind korrekt?
- ☐ Es gibt nur einen Siliziumhersteller: STMicroelectronics.
 - ☐ Auf dem Silizium sind immer auch Programm- und Datenspeicher integriert.
 - ☐ Sie basieren auf einer CISC-Architektur von ARM.
 - ☐ Die CPU taktet so schnell, dass es einen Cache braucht.
-
- c) Welche Aussagen zu Cortex-Mx CPUs sind korrekt?
- ☐ Sie enthalten immer eine ALU.
 - ☐ Sie enthalten eine seriellen Schnittstellen.
 - ☐ Sie enthalten zwei Core-Register.
 - ☐ Sie enthalten einen Program Counter (PC).
-
- d) Welche Aussagen zur MMU sind korrekt?
- ☐ Die MMU übersetzt physikalische in virtuelle Adressen.
 - ☐ Die Page-Tables werden im RAM abgelegt.
 - ☐ Der TLB ist im RAM abgelegt.
 - ☐ Die MMU optimiert die Zugriffsgeschwindigkeit auf den Speicher.
-
- e) Welche Aussagen zum Cache sind korrekt?
- ☐ Der Cache ist ein schneller Zwischenspeicher für Daten und Programmcode.
 - ☐ Ein "virtueller" Cache liegt zwischen MMU und Speicher.
 - ☐ Bei einem Cache-Miss wird eine Cache-Line vom Arbeitsspeicher in den Cache geladen.
 - ☐ Im Cache werden die Core-Register der CPU zwischengespeichert.
-
- f) Welche Aussagen zur Floating-Point Unit sind korrekt?
- ☐ Die FPU kann nur Floating-Point Operationen durchführen.
 - ☐ Die FPU unterstützt die ALU bei Multiplikation und Division von Integer-Zahlen.
 - ☐ Die C-Anweisung "sin(x) + 2.3F" wird im Single Precision Format (32-Bit) ausgeführt.
 - ☐ Floating-Point Berechnungen werden in Hardware mit einer FPU oder in Software mit Hilfe einer Library durchgeführt.
-

Aufgabe 2: Multiple Choice Assembler (je 0.5 Punkte, total 3 Punkte)

-
- a) Wie viele Assembler-Instruktionen werden benötigt, um den Wert einer Variablen im RAM zu löschen ("var= 0;", wobei var eine 32-Bit Variable ist)?
- ☐ 1
 - ☐ 2
 - ☐ 3
 - ☐ 4
-
- b) Gegeben ist eine Assembler-Subroutine, welche aus Sicht der Programmiersprache C folgende Deklaration hat: `"int myFoo(char *v1, char v2, int v3, short v4, int v5);"` Welche unten aufgeführten Aussagen sind für einen Cortex-M7 korrekt?
- ☐ v2 wird "by reference" im Register r1 übergeben.
 - ☐ v1 wird "by reference" im Register r0 übergeben.
 - ☐ v5 wird "by value" über den Stack übergeben.
 - ☐ v3 und v4 werden über den Stack in umgekehrter Reihenfolge übergeben.
-
- c) Das Register r4 habe den Wert 0x20001000. Welchen Wert hat es nach Ausführung der Instruktion `"STMIA r4!, {r0, r3}"`?
- ☐ 0x20001002
 - ☐ 0x20001008
 - ☐ 0x20000FFE
 - ☐ 0x20000FF8
-
- d) Welche der unten aufgelisteten Assembler-Instruktionen führen einen Rücksprung aus einer Subroutine aus?
- ☐ POP {r5,pc}
 - ☐ MOV lr,pc
 - ☐ POP {r5,lr}
 - ☐ PUSH {pc}
-
- e) Welche Aussagen sind für einen ARM Cortex-M7 und Subroutinen korrekt?
- ☐ r1 wird vor dem Start der Subroutine immer automatisch auf dem Stack gerettet.
 - ☐ r4 muss eingangs der Subroutine gerettet werden, sofern r4 in der Subroutine modifiziert wird.
 - ☐ r3 muss immer eingangs der Subroutine gerettet werden.
 - ☐ r0 wird für den Rückgabewert verwendet und muss deshalb nicht gerettet werden.
-
- f) Die Variable "var" habe den Wert 0x12345678. Was hat die Variable nach Ausführung der nachfolgenden Assembler-Instruktionen für einen Wert?
- ```
LDR r0,=var @ kopiere Adresse Variable var in r0
LDR r1,[r0]
BIC r1,r1,#(1<<3)
STR r1,[r0]
```
- ☐ 0x12345678
  - ☐ 0x12345078
  - ☐ 0x12345670
  - ☐ 0x12345673
-





```

... <-- Momentaufnahme vor Aufruf Aufgabe3d
MOV r0,#5 @ first parameter
BL Aufgabe3d @ call subroutine
... <-- naechste Assembler-Instruktion: Adresse 0x0800039A

```

Aufgabe3d ist in Assembler wie folgt implementiert:

Aufgabe3d:

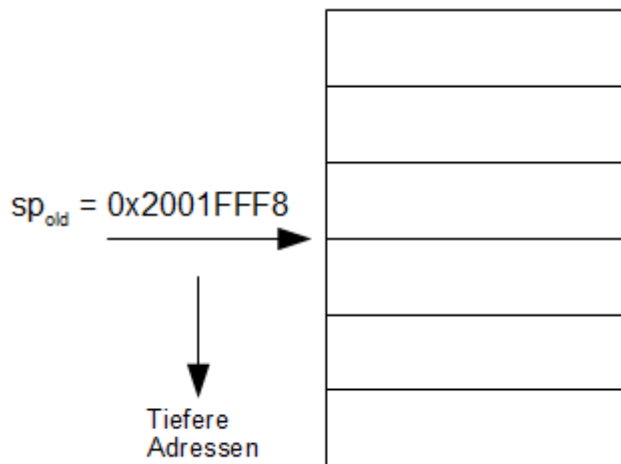
```

 PUSH {lr} @ push lr on stack
 ... <-- Stack und Registerwerte innerhalb von Aufgabe3d
 POP {pc} @ return

```

Der Stackpointer hat vor dem Aufruf von Aufgabe3d folgende Werte: 0x2001FFF8

Tragen Sie den Inhalt des Stacks innerhalb der Subroutine Aufgabe3d in nachfolgender Grafik ein. Geben Sie an, welche Werte die Register r0, lr und sp innerhalb der Subroutine Aufgabe3d haben.



Registerwerte innerhalb von Aufgabe 3d:

r0: 0x

lr: 0x

sp: 0x



#### Aufgabe 4: Assembler-Subroutine getCharPosition() (4 + 2 = 6 Punkte)

In Assembler soll eine Subroutine geschrieben werden, die in einem String (als Parameter übergeben) den ersten Buchstaben 'u' sucht und dessen Position zurück gibt. Der erste Buchstabe des Strings hat Position 1, der zweite Buchstabe Position 2 usw. Falls der Buchstabe nicht gefunden wird, wird 0 (Null) zurück gegeben. Diese Subroutine hat folgende Deklaration:

```
unsigned long getCharPosition(char * s);
```

- a) Subroutine in Assembler:

Implementieren Sie die **Subroutine “getCharPosition” in Assembler**. Parameter und Rückgabewert müssen gemäss AAPCS übergeben werden. Bitte kommentieren Sie den Assembler-Code für “getCharPosition”, die Kommentare werden mit 1 Punkt bewertet.

This image shows a full page of blank graph paper. The background is a very light gray, and it is covered by a precise grid of thin, medium-gray lines. The grid consists of small, identical squares that extend across the entire area of the page, leaving no margins or other markings.

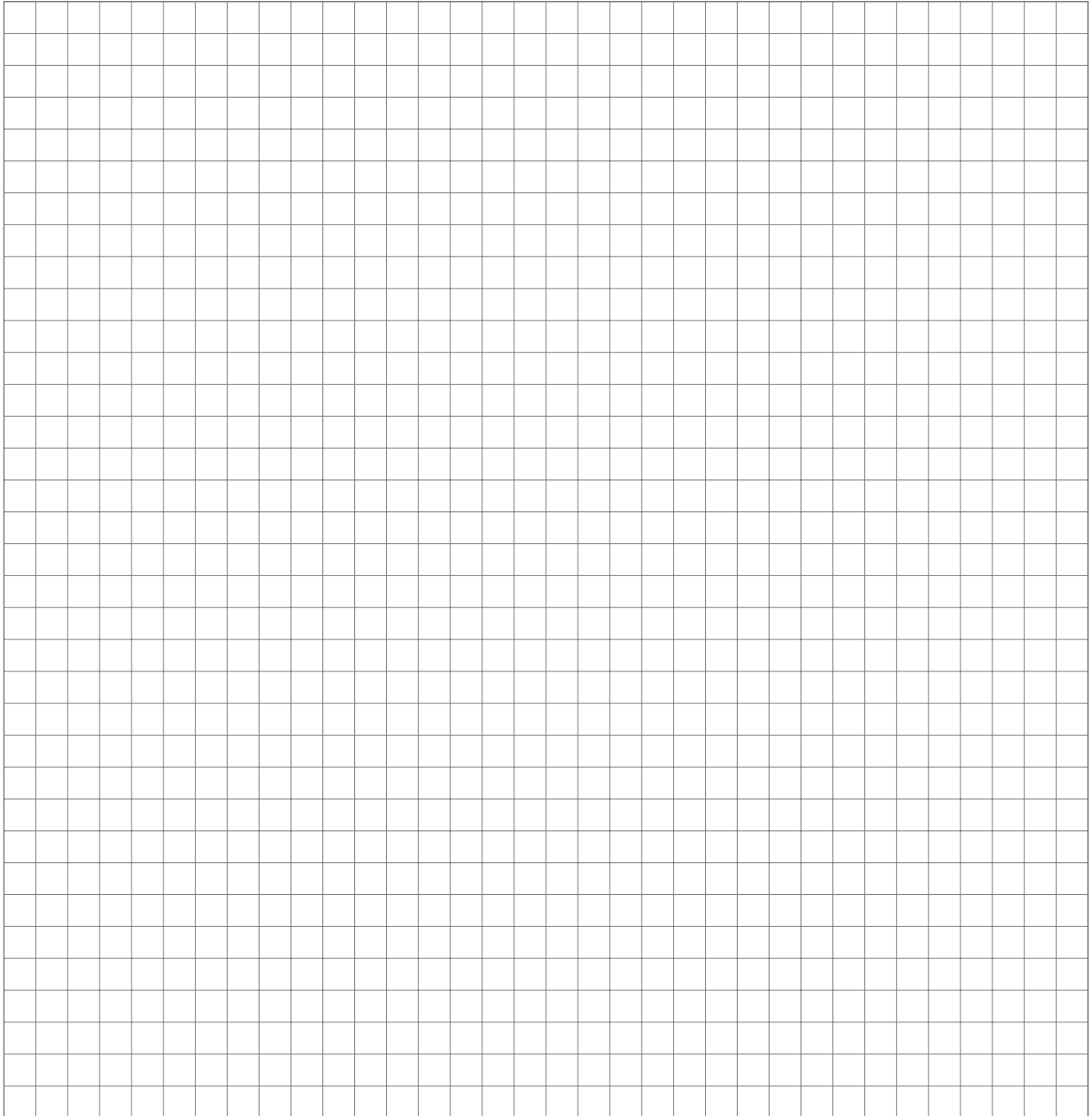
Für die Korrektur:

Kommentare:

Funktion:

b) Testprogramm in Assembler:

Schreiben Sie ein **Testprogramm in Assembler**, in welchem Sie einen String "str1" initialisiert auf "Burgdorf" anlegen und anschliessend die Subroutine "getCharPosition" aufrufen. Der Rückgabewert von "getCharPosition" soll in der Variablen "res" gespeichert werden.



Für die Korrektur:

Funktion: