Test

Hardwarenahe Softwareentwicklung

6. Mai 2022

Name: _____ **Klausur-ID: 10**

Aufgabe	**1**	**2**	**3**	**4**	**∑**
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.
- Zuzlässige 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

Aufgabe 3: Kurzfragen (bitte Rückseite beachten!) (2 + 2 + 2 + 2 = 8 Punkte)

Beantworten Sie die folgenden Kurzfragen:

a) MMU Speicherbedarf
Wie gross ist eine Page-Table einer 32-Bit CPU in Kilobyte, wenn eine Page 1 Megabyte gross ist? Nehmen Sie als Vereinfachung an, dass eine Page-Table Entry 32 Bit gross ist. Beschreiben Sie auch den Lösungsweg.

32 bit Adressbus → 4 Giga Adressen → 4k pages.

1 page entry hat 4 Byte → page table ist 16 kByte gross.

1 Punkt für Anzahl pages (4 kByte)

1 Punkt für Speichergrösse (16 kByte)

b) Cache
Wie gross ist das Cache Memory, wenn der Data-Index 4 Bit und der Set-Index 9 Bit gross ist? Beschreiben Sie auch den Lösungsweg.

Data-Index 4 Bit → Cache-Line hat 16 Byte

Set-Index 9 Bit → es hat 512 Cache Lines

→ total 16 Byte * 512 = 8 kByte

1/2 Punkte für Anzahl Bytes pro Cache-Line (16)

1/2 Punkte für Anzahl Cache-Lines (512)

1 Punkt für Speichergrösse (8 kByte)

c) Assembler-Direktiven
Mit Hilfe der Makros .rept, .set und .word soll eine Tabelle mit 10 Word-Konstanten erzeugt werden, deren Elemente folgende Werte annehmen: 1, 2, 4, 8, 16 usw. Die Tabelle soll mit dem Label “tab“ im .text-Segment angelegt werden.

```
    .align                  ..................................... 1/4
    .text                   ..................................... 1/4

    # Erzeugen der Tabelle fuer alle  10 Byte-Werte.
    .set i,1                @ aktueller Wert ..................... 1/4
tab:                        ..................................... 1/4
    .rept 10                @ Fuer 16 Tabellenwerte ............. 1/4
    .word i                 @ Speicher allozieren und Wert setzen .. 1/4
    .set  i,i*2             @ naechsten Wert vorbereiten ........... 1/4
    .endr                   ..................................... 1/4
```

d) Stack:
In Assembler wurde die Subroutine “Aufgabe3d“ implementiert, welche aus einer anderen Assembler-Subroutine aufgerufen wird. Der Code für den Aufruf lautet :

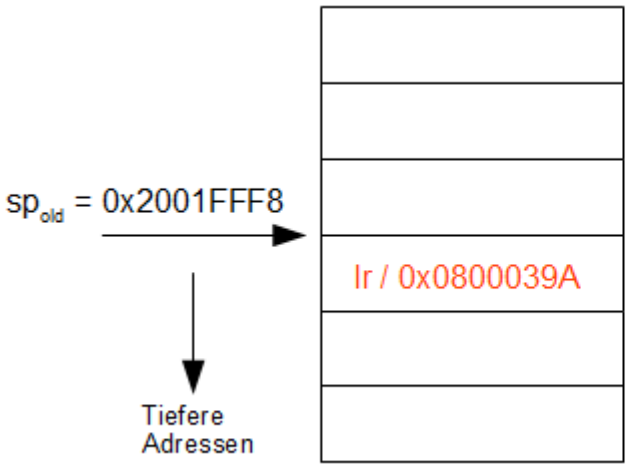
```
    ...  <-- Momentaufnahme vor Aufruf Aufgabe3d
    MOV r0,#5       @ first parameter
    BL  Aufgabe3d   @ call subroutine
    ...  <-- naechste Assembler-Instruktion: Adresse 0x0800039A
```

Aufgabe3d ist in Assembler wie folgt implementiert:

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

Der Stackponter 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: 5

lr: 0x0800039A

sp: 0x2001FFF4

Je 1/2 Punkt für Stackeintrag und die einzelnen Register

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.

```
    .text                           @ section text (executable code)

/**
* @brief   Searches the char position in a string.
*
* Reads each character of a string until the specified char is found.
* If char is not found, 0 will be returned.
* The position of the char is calculating using address of found character
* (r1) and start address of string.
*
* @param   r0: String base address
* @return  r0: Position of char
* @remark  r1 used to point to actual char in string
*          r2 holds actual char
*           r3 holds start address
*/
getCharPosition:
    MOV     r1,r0                   @ r1 points to start character
    MOV     r3,r0                   @ r3 holds start adress to calculate position
    MOV     r0,#0                   @ return value if NULL-String
getCharPosition_loop:
    LDRB    r2,[r1],#1              @ load actual character, increment pointer r1
    CMP     r2,#0                   @ end of string?
    BEQ     getCharPosition_end     @ yes --> end, char not found, return 0
    CMP     r2,#'x'                 @ check if character is 'u'
    BNE     getCharPosition_loop    @ no --> next character
    SUB     r0,r1,r3                @ yes --> char found, position is difference of
       addresses
getCharPosition_end:
    MOV     pc,lr                   @ return
```

Punkte:
Kommentare 1 Punkt
Funktion 3 Punkte
Grössere Fehler -1 Punkt

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 Rückgabwert von “getCharPosition“ soll in der Variablen “res“ gespeichert werden.

```
    .data
    .align
str1:
    .asciz  "Burgdorf"

    .text
/**
* @brief       main subroutine
*
* The main subroutine calls subroutine count_e twice, each time with
* a differnt string. The return value of count_e is not processed. Use
* the debugger to check the values.
*
* @param       none
* @return      none
*/
main:
    /**
    * endless loop
    */
loop:
    LDR     r0,=str1                @ load r0 (argument) with string base address
    BL      getCharPosition         @ call subroutine
    B       loop
```

Punkte:
Variable str1 1/2
main und loop 1/2
Parameter in r0 1/2
Aufruf Subroutine 1/2