CT1 Übungsaufgaben
Datenübergabe / Schnittstelle zu Hochsprachen

Aufgabe 1

Gegeben ist folgernder C-Code:

```
uint32_t logical_and(uint32_t a, uint32_t b, uint32_t c)
{
  return a & b & c;
}


int32_t main(void)
{
  uint32_t result;
  uint32_t x = 0x11223344;
  uint32_t y = 0xFFFF0000;
  uint32_t z = 0x33661122;

  result = logical_and(x, y, z);
}
```

Beim Start des Programmes wird die Variable x in R4, y in R5 und z in R6 abgelegt. Die Variable result wird in R7 abgelegt.

1. Welche Schritte führt der Caller (main) vor dem Aufruf der Funktion logical_and() durch? Wie werden die Parameter übergeben?

   Die Variablen in den Registern R4 bis R6 werden nach R0 bis R2 kopiert und so der Funktion übergeben. (R6 => R2, R5 => R1, R4 => R0)

2. Wir gibt die Funktion logical_and() den Rückgabewert zurück?

   Der Rückgabewert wird via R0 zurückgegeben.

3. Welche Operation führt der Call nach dem Aufruf der Funktion logical_and() durch?

   Der Rückgabewert wird von R0 nach R7 kopiert

Aufgabe 2

Gegeben ist das folgende C-Programm:

```
#include <utils_ctboard.h>

void swap_bad(int32_t c, int32_t d)
{
  /*WARNING: This code does not work*/
  int32_t temp = c;
  c = d;
  d = temp;
}

int32_t main(void)
{
  int32_t a = 3, b = 5;

  swap_bad(a,b);

  write_word(0x60000300, a);
}
```

a. Die Swap-Funktion swap_bad() lässt sich ohne Fehler kompilieren und ausführen. Nach ihrem Aufruf in main() hat die Variable a jedoch immer noch den Wert 3. Erläutern Sie anhand der Calling Convention wo das Problem liegt.

Beim Aufruf der Funktion swap_bad() werden die übergebenen Parameter in die Register R0, R1 kopiert und in der Funktion auch korrekt verändert. Jedoch werden die Änderungen nicht an den Caller zurückgegeben. Daher sind die Änderungen in main() nicht sichtbar.

b. Schreiben Sie eine Funktion swap_good() die diesen Fehler behebt und korrekt funktioniert

```
void swap_good(int32_t *c, int32_t *d){
  int32_t temp = *c;
  *c = *d;
  *d = temp;
}

int32_t main(void){
  int32_t a = 3, b = 5;

  swap_good(&a, &b);

  write_word(0x60000300, a);
}
```

Aufgabe 3

Gegeben sei das folgende C-Programm:

```
int32_t fakultaet_recursive(int32_t n)
{
  if(n < 2){
    return 1;
  }
  else{
        return n * fakultaet_recursive(n-1); // Break Point set here
  }
}

int32_t main(void)
{
  int32_t n = 20;

  int32_t  result = fakultaet_recursive(n);
}
```

Das C-Programm berechnet die Fakultät von 20 und verwendet dazu Rekursion. Dabei ruft sich die Funktion selbst wieder auf. Dies wird so lange wiederholt, bis die Abbruchbedingung (n < 2) erreicht wird.

Wenn das Programm zum zweiten Mal beim Breakpoint in der Funktion fakultaet_recursive() gestoppt wird, ist das Stackframe der main()-Funktion und der aufgerufenen fakultaet_recursive()-Funktionen 16 Byte gross. Die Variable n wird in einem Register gespeichert.

Wie gross wird das Stackframe beim Ausführen des Programmes maximal?

Da die Variable n in einem Register gespeichert ist und das Stackframe der main()-Funktion somit 0 ist, wird bei jedem Aufruf der Funktion fakultaet_recursive() das Stackframe um 8 Byte (R4 und LR) erhöht. Nach zwanzig Aufrufen ist das Stackframe also 160 Byte gross.