Stackanalyse

main: //Hauptprogramm

bl startup

ldr R1,=GPIOB

bl unterprogramm

ldrb R0,[R1,IDR] //Eingabe von GPIOB.IDR

ldr R1,=GPIOC

bl unterprogramm

strb R0,[R1,ODR] //Ausgabe auf GPIOC.ODR

schleife:

b schleife

unterprogramm: //Wichtiges Unterprogramm verwendet unbedingt Register R1

mov R1,#0x110

bx lr

.end

Obiges Programm muss unbedingt die Zahl von GPIOB.IDR einlesen und auf die an GPIOC.ODR angeschlossenen LEDs ausgeben. Sonst passiert ein großes Unglück.

Aufgabe 1: Analysieren Sie ob das so funktionieren kann.

Aus unerfindlichen Gründen gelten folgende Randbedingungen:

* Das darf Hauptprogramm nicht verändert werden.
* Der Entwickler des Unterprogramms besteht auf die Verwendung von Register R1 und er ist der Sohn des Firmeninhabers

Aufgabe 2: Ein Entwickler schlägt folgende Lösung vor. Prüfen Sie die Lösung mit dem Debugger, füllen Sie hierzu die Tabelle unten aus.

Leicht verändertes Programm

main: //Hauptprogramm

bl startup

ldr R1,=GPIOB

bl unterprogramm

ldrb R0,[R1,IDR] //Eingabe von GPIOB.IDR

ldr R1,=GPIOC

bl unterprogramm

strb R0,[R1,ODR] //Ausgabe auf GPIOC.ODR

schleife:

b schleife

unterprogramm: //Wichtiges Unterprogramm verwendet unbedingt Register R1

**push {R1}**

mov R1,#0x110

**pop {R1}**

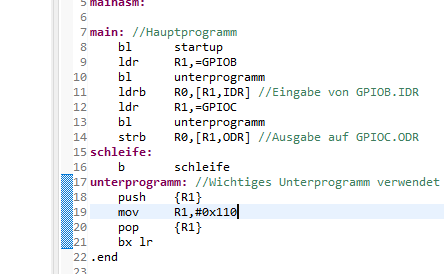
bx lr

.end

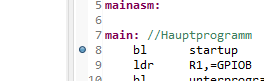
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Programm | Weiter mit | => | R13 SP | R1 | 0x20013ff4 |
| ldr R1,=GPIOB | Step in F5 | => | 0x20013ff8 | - |  |
| bl unterprogramm | Step in F5 | => |  |  |  |
| push {R1} | Step in F5 | => |  |  |  |
| mov R1,#0x110 | Step in F5 | => |  |  |  |
| pop {R1} | Step in F5 | => |  |  |  |
| bx lr | Step in F5 | => |  |  |  |
| ldrb R0,[R1,IDR] | Step in F5 | => |  |  |  |
| ldr R1,=GPIOC | Step in F5 | => |  |  |  |
| bl unterprogramm | Step in F5 | => |  |  |  |
| push {R1} | Step in F5 | => |  |  |  |
| mov R1,#0x110 | Step in F5 | => |  |  |  |
| pop {R1} | Step in F5 | => |  |  |  |
| bx lr | Step in F5 | => |  |  |  |

Anleitung:

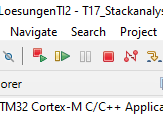
1. Programmieren Sie obiges Programm
2. Setzen Sie auf **bl startup** einen Breakpoint



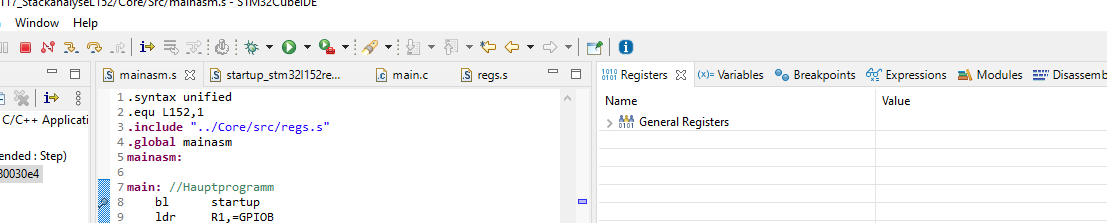
Rechtsklick und Toggle Breakpoint



1. Starten Sie das Programm im Debugmode mit Debug As
2. Programm starten mit **resume**



1. Window -> Show View -> Registers



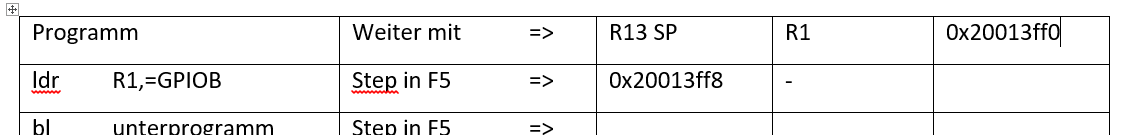
Registers aufklappen

Aufgabe 4: Beantworten Sie folgende Fragen:

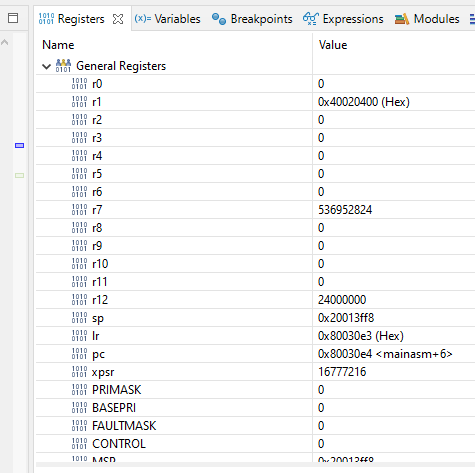
1. Welche Wirkung hat die Instruktion push {R1} auf den Stackpointer SP (R13)  
   und welche Aktion führt sie mit dem Inhalt von Register R1 aus?
2. Welche Wirkung hat die Instruktion pop {R1} auf den Stackpointer SP (R13)

und welche Aktion führt sie mit dem Inhalt von Register R1 aus?

1. Auf Hexadezimale Anzeige umstellen



4 weniger



Rechtsklick  
Numberformat Hex

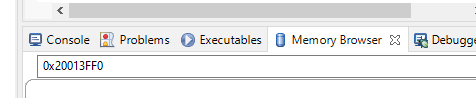
Rechtsklick  
Numberformat Hex

1. Wert des Stackpointers SP (R13) (hier 0x20013ff8) ablesen und in Tabelle eintragen

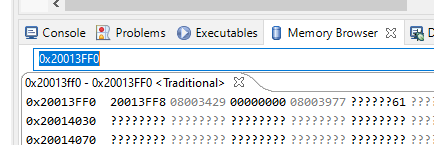
0x20013F4

Hierher übertragen und enter-Taste

1. Window -> Show View -> Memory Browser



0x20013F4



RAM – Speicher des Stacks wird angezeigt

1. Programm mit Step Into (F5) im Einzelschrittbetrieb durchtakten und bei jedem Schritt werte in der Tabelle protokollieren

Aufgabe 5: Untersuchen Sie in gleicher Weise die Instruktionen: push {R0,R1,R2} und pop {R0,R1,R2}

Hinweis: Dazu muss im Memory Browser eine Adresse ausgewählt werden, die um nochmals 8 niedriger ist.