Aufgabenblatt 3 Modulare Programmierung

1 Unterprogrammaufruf (activ waiting delay)

Implementieren Sie eine Funktion für eine primitive Wartezeit (Abbildung 1). Da die Wartezeit in einer temporären Variablen (Register) runtergezählt wird, ist dieses Register am Anfang dieser Funktion zu speichern. Danach kopieren Sie den konstanten Parameter („***.equ***“) in Ihrem temporären Register. Nun müssen Sie dieses Register bis auf 0 runterzählen. Am Ende wiederum ist der Wert des temporären Registers auf den Wert vor dem Betreten der Funktion wiederherzustellen.
Erweitern Sie das Programm aus der Aufgabe 2.3 so, dass in jedem Schleifendurchlauf am Ende Ihre neue ***delay*** Funktion einmal aufgerufen wird. Bitte verwenden Sie in Ihrer Lösung keine Pseudobefehle für den Stack zugriff.

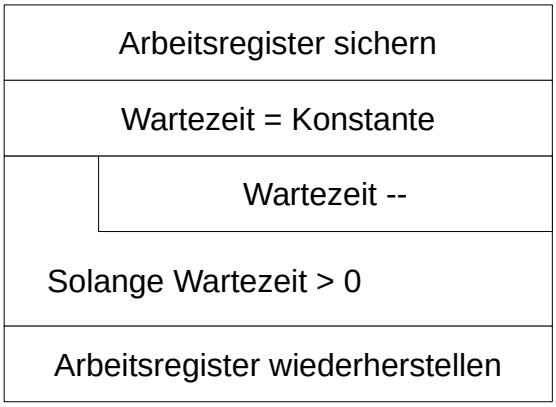

Abbildung 1: Funktion ohne Parameter

Skript:
1.12 Stack
1.12.1 Funktionsweise des Stacks (Daten ablegen)
2.5 Stack Operationen
2.2.5.4 Mehrfaches Laden mit Postdekrement
2.4.3 Mehrfaches Speichern mit Postdekrement
2.8.3 Bedingter Sprung mit Link (Funktionsaufruf)
2.8.4.1 Anwendungsfall – Subroutinen, Funktionen

Nur zur Verständnis / Beispiel. Aus didaktischen Gründen sind diese Pseudobefehle Befehle noch nicht erlaubt:
2.5.1 „Push“ auf dem Stack speichern
2.5.2 „Pop“ vom Stack holen
Statt dessen verwenden sie bitte die entsprechenden Load ans Store Befehle (Klausur relevant)

Tipps:
- - Skript - <strg>+<f> „d...y“
- Zum testen der Verzögerungszeit, vor allem wenn diese in dem Einzelschrittverfahren im Debugger getestet wird, empfählen Sich die Werte von 2 oder 3. 99999 oder ähnlich dauern eine Weile…..

2 Unterprogrammaufruf mit Parametern (Delay)

Feste Wartezeiten sind ein wenig unflexibel, also modifizieren Sie die Aufgabe 1 so, dass die Verzögerungszeit jetzt über einen Parameter an die Funktion übergeben wird (Abbildung 2 und 3). Wenn Sie den Parameter vor dem Funktionsaufruf immer wieder initialisieren, so können Sie auf die Stackoperationen verzichten (Abbildung 3). In diesem Fall müssen Sie dann mit der ***bx*** Anweisung zurückspringen. Auch hier verwenden Sie bitte die Aufgabe 2.3 um die Funktion zu testen. Diesmal können Sie die beiden Pseudobefehle ***push*** und ***pop*** verwenden.

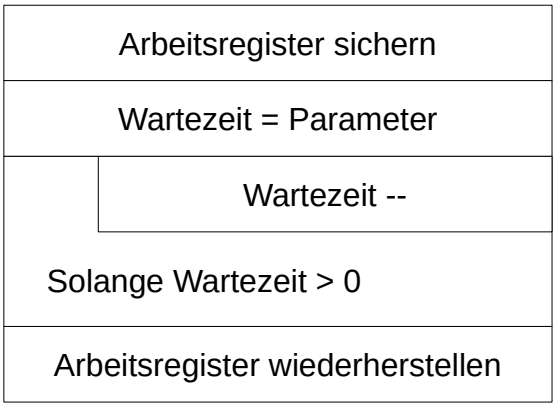

Abbildung 2: Funktion mit Parameter

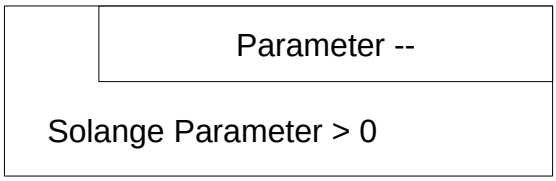

Abbildung 3: Funktion mit Parameter

Skript:
2.8.4.2 Anwendungsfall – Parameterübergabe
2.8.4 Sprung mit Austausch (bx)

3 Unterprogrammaufruf mit Parameterübergabe über dem Stack

Die Compiler machen das für gewöhnlich ein wenig anders. Vor allem, wenn mehrere Parameter an eine Funktion übergeben werden müssen, kann man dafür nicht viele der „wertvollen“ Register für eine Parameterübergabe „verbraten“. So wird die Parameterübergabe in beide Richtungen oft über den Stack abgewickelt.

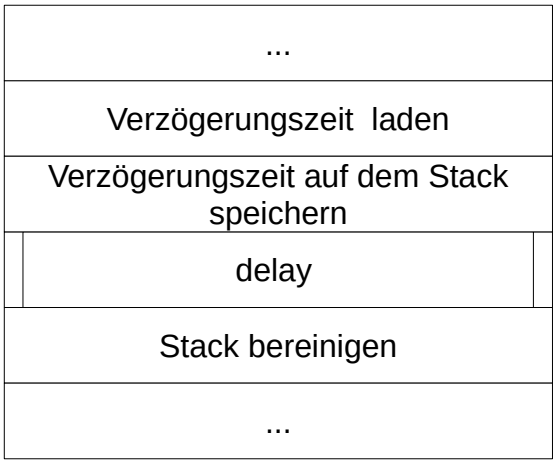

Abbildung 4: Hauptprogramm

Modifizieren Sie die Aufgabe 2 jetzt so, dass der Wartezeitparameter über den Stack an die Funktion übergeben wird. In der Abbildung 4 ist ein kleiner Auszug ,mit der Änderung die Sie an Ihrem Hauptprogramm durchführen müssen ,zu sehen. Bevor die ***delay*** Funktion ausführt wird, müssen Sie dafür sorgen, dass der Parameter auf den Stack gelegt wird. Selbstverständlich muss auch der Stackpointer um die Größe der dort abgelegten Variable(n) verschoben werden (Post--krement sagt „hallo“). Beides kann mit einem geeigneten Befehl in einer Operation durchgeführt werden. Ist der Parameter abgelegt, so kann die Funktion aufgerufen werden. Direkt nach dem Verlassen der Funktion muss der Stack wiederum bereinigt werden, entweder mit einem ***load*** oder mit einem entsprechenden ***add/sub*** Befehl.

Die Funktion selbst muss auch entsprechend angepasst werden. Der Parameter kann jetzt über indirekte Adressierung vom Stack direkt geladen werden. Bei der Berechnung der Adresse bzw. des Indexes ist die Größe der am Anfang der Funktion auf dem Stack abgelegten Daten zu berücksichtigen. Der Ladezugriff selbst verändert den Stackpointer nicht, da dieser noch für das Wiederherstellen des Anfangswertes des verwendeten temporären Registers notwendig ist.

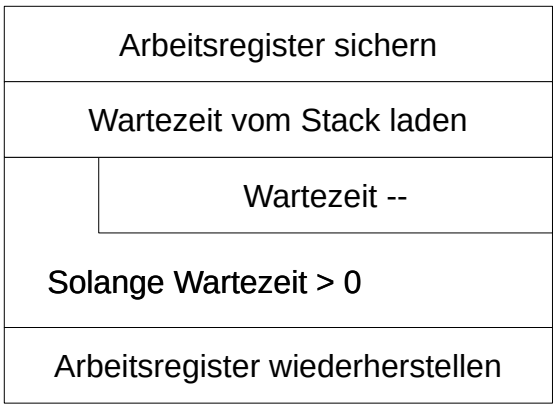

Abbildung 5: Funktion

Skript:
4.13 Parameterübergabe über dem Stack

4 Unterprogrammaufruf mit Übergebe von mehreren Parametern - Division

Wir wollen das mit dem Parameterübergabe nochmals etwas üben. Da wir die Multiplikation bereits implementiert haben, beschäftigen wir uns jetzt mal zu Abwechslung mit der Division. Der einfachste Algorithmus, den wir hier verwenden wollen, ist wenn einfach der Divisor solange von dem Dividend abgezogen wird, bis es „nicht mehr weiter geht“. In der Abbildung 6 ist dieses Verfahren bildlich dargestellt. Ist der Restwert 0 dann bricht die Addition ab. Leider lässt sich das nicht so direkt umsetzten, da in den meisten Fällen ein Restwert !=0 bleibt. Deswegen geht der Algorithmus einen Schritt weiter und prüft erst auf einen negativen Rest (Abbildung 7). Wird ein negativer Rest festgestellt, so muss einerseits das Ergebnis um eins vermindert werden und der Rest in dem Dividenten wiederhergestellt werden.

Dividend	Divisor	Rest	
8	-2	6	1
6	-2	4	2
4	-2	2	3
2	-2	0	4

Abbildung 6: Division ohne Rest

Divident	Divisor	Rest	
7	-2	5	1
5	-2	3	2
3	-2	1	3
1	-2	-1	4
-1	+2	1	3

Abbildung 7: Division mit Rest

Implementieren Sie eine Funktion, die zwei gegebene Zahlen dividiert und das Ergebnis sowie den Rest zurückliefert. Sie müssen nur einen der beiden folgenden Ansätze umsetzen.

4.1 Parameterübergabe über Register

In der einfacheren Implementierung werden die benötigten Parameter vor dem Aufruf der Funktion in die allgemeinen Register geladen, so dass diese nach dem Aufruf direkt in der Prozedur verwendet werden können (Abbildung 8). Die Ergebnisse der Funktion werden wiederum von dieser direkt in zwei weitere Registern zurückgeliefert.

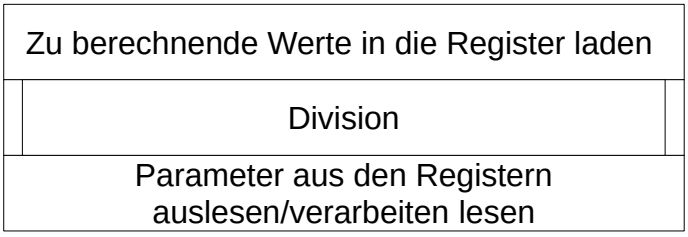

Abbildung 8: Funktionsaufruf mit

Die Funktion selbst ist, wie bereits besprochen, ganz einfach (Abbildung 9). Zuerst werden alle Register, die für temporäre Werte verwendet werden, auf dem Stack gespeichert. Im nächstem Schritt folgt die Initialisierung der Laufzeitvariablen mit den entsprechenden Startwerten. Nun wird in der Schleife, solange der Dividend größer oder gleich 0 ist, der Divisor vom Dividenden abgezogen. Dabei wird bei jeder Subtraktion das Ergebnis um eins erhöht. Ist der Dividend negativ geworden, so ist das Ende der Schleife erreicht. Dies bedeutet aber auch, dass hier bereits eine Subtraktion zu viel gemacht worden ist, so muss das Ergebnis korrigiert und der Rest wiederhergestellt werden, indem ein Schleifendurchlauf „zurückgenommen“ wird.

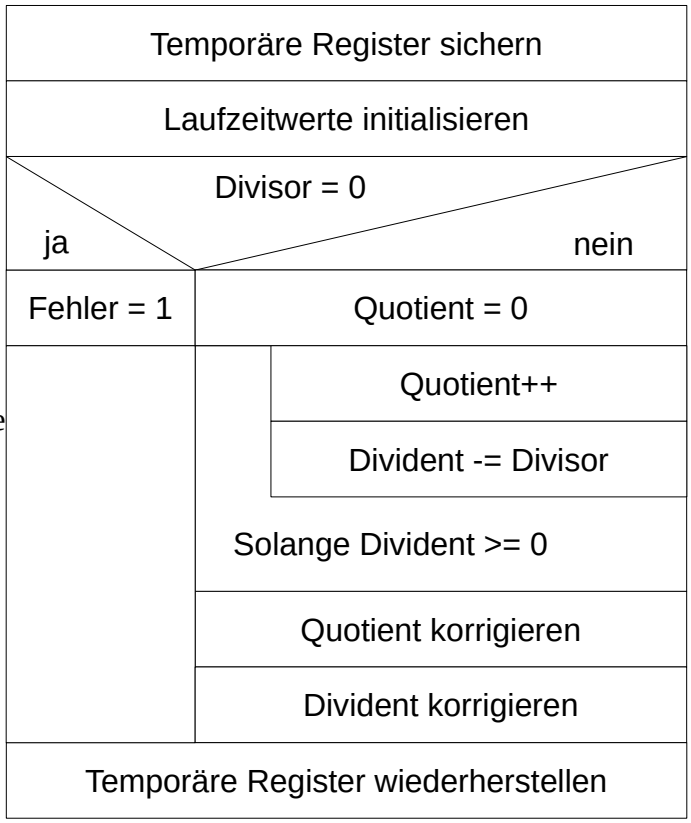

Abbildung 9: Ansatz mit Parameterübergabe über Register

4.2 Ansatz 2 – Parameterübergabe über den Stack

Die zweite Möglichkeit die Aufgabe zu lösen ist so vorzugehen, wie es ein Compiler machen würde, wenn ihm freie Register ausgehen würden. Hierbei werden die Parameter,wie in der Aufgabe 3.3, nur über den Stack übergeben. Das Hauptprogramm ist hier etwas erweitert, da es sich jetzt zuerst zusätzlich noch um das Ablegen der Variablen auf dem Stack und die Verschiebung des Stackpointers um den Parameterdatenblock kümmern muss. Erst dann kann die Funktion selbst gestartet werden. Analog dazu muss nach der Rückkehr aus der Funktion der Stack wieder aufgeräumt werden.

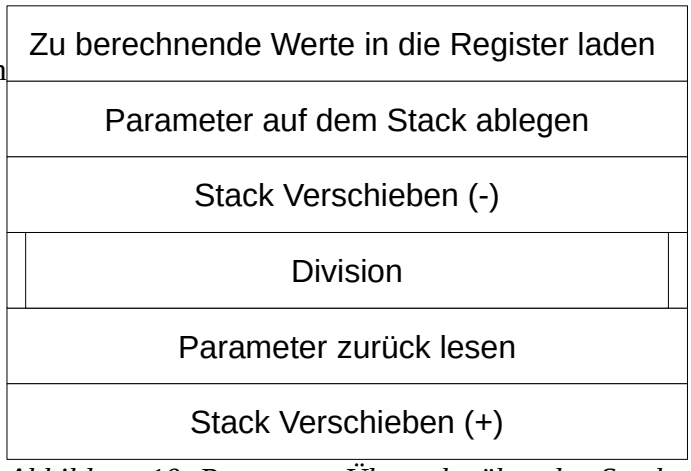

Abbildung 10: Parameter Übergabe über den Stack - Hauptprogramm

Die Funktion selbst unterscheidet sich vom vorhergehenden Ansatz nur dadurch, dass hier als erstes, nach dem Speichern der verwendeten temporären Register, die Parameter selbst vom Stack geholt werden. Des weiteren müssen zum Schluss, bevor die Ursprungswerte der temporären Register aus dem Stack geholt werden, die Ergebnisse auf dem Stack zurückgeschrieben werden. Da die Parameter nicht oben auf dem Stack liegen, darf der Stackpointer bei diesen beiden Operationen nicht verschoben werden. Der Zugriff auf die Parameter erfolgt deswegen über Stackpointer relative Indexe.

Skript:
2.8.4.3 Autonomie der Funktionen

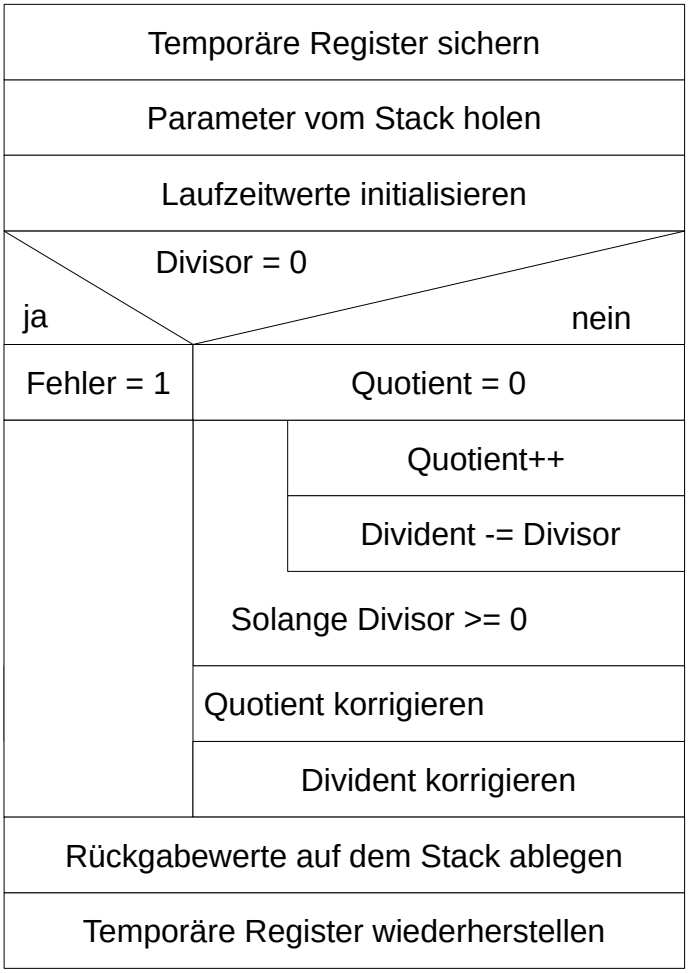

Abbildung 11: Parameterübergabe über den Stack - Funktion