Aufgabenblatt 6: Assembler-Programmierung 2

Auf diesem Übungszettel sollen Sie versuchen, die Funktion eines Assembler Programmes zu bestimmen und ein Unterprogramm in ein bestehendes Assember-Programm zu integrieren. Folgende Befehle sollten Ihnen aus der Dokumentation von SPIM bekannt sein:

Einige Befehle

`lw $t0, address`	Läd eine 32-Bit Zahl (word) von der Speicherstelle *address* in das Register *$t0*
`li` *$t0*, *c*	Lade Konstante *c* in das Register *$t0* (Pseudobefehl)
`la` *$t0*, *address*	Lade *adress* (**Vorsicht:** Nicht den Inhalt der Adresse sondern die Adresse selbst) in das Register *$t0* (Pseudobefehl)
`addi` *$t0*, *$t1*, *c*	Addiert die Konstante *c* auf das Register *$t1* und legt das Ergebnis in *$t0* ab
`div` *$t0*, *$t1*	Teilt den Inhalt zweier Register (hier: *$t0*/*$t1*. Quotient wird im Register *lo* und der Rest in Register *hi* gespeichert
`mfhi` *$t0*	Transportiert den Inhalt von *hi* in das Register *$t0*
`beq` *t0*,t1, label	Bedingter Sprung zur Speicherstelle *label* wenn der Inhalt der beiden Register *$t0* und *$t1* gleich ist
`jal` *label*	Unbedingter Sprung an den Befehl bei Adresse *label*. Speichert die Adresse des nach *jal* folgenden Befehls in Register *$ra* (*$31*)
`j` *$label*	Unbedingter Sprung zum Befehl an Adresse *label*
`jr` *$ra*	Unbedingter Sprung zum Befehl dessen Adresse in Register *$ra* steht
`syscall`	Aufruf einer Kernel-Routine (Pseudobefehl)

Register

Registername	Nummer	Gebrauch
zero	0	Konstant 0
at	1	Reserviert für den Assembler
v0, v1	2, 3	Register für Rückgabewerte einer Funktion
a0 - a3	4 - 7	Register für Argumente bei Funktionsaufruf
t0 - t7, t8, t9	8 - 15, 24, 25	Temporäre Register (werden bei Funktionsaufrufen nicht gesichert)
s0 - s7	16 - 23	Geschützte temporäre Register (werden bei Funktionsaufrufen gesichert)
k0, k1	26, 27	Reserviert für den Kernel
gp	28	Zeiger auf den globalen Bereich
sp	29	Zeiger auf den Stack
fp	30	Zeiger auf den Frame
ra	31	Rücksprungadresse (Wird bei Funktionsaufrufen benötigt)

Kernel-Routinen

Mit dem Befehl syscall werden spezielle Kernel-Routinen des Simulators aufgerufen. Den Routinen wird im Register $v0 ein Kommandokode und im Register $a0 ein Parameter übergeben.

```
$v0  $a0
1    Adresse  Der Integer-Wert der gegebennen Adresse wird ausgegeben
4    Adresse  Der nullterminierte Text ab der Adresse wird ausgegeben
```

Falls Ihnen die Funktionsweise des einen oder anderen Befehles nicht klar sein sollte schauen Sie bitte in die Dokumentation des SPIM-Simulators. Diese finden sie unter:

http://wwwhni.upb.de/eps/uni/courses/gra/uebung/spimemulator/mipssim.php3

Aufgabe 1: Puzzel

Folgendes Assembler-Programm ist gegeben:

```
        .data
        .align   2

NUM:    .word    12 #NUM must be >=3
STR1:   .asciiz  "not successfull Number : "
STR2:   .asciiz  "successfull Number: "

        .text
        .align   2
        .globl   main

main:   lw   $a1, NUM        # read the Number to test
        li   $a2, 2          # start test with 2
        li   $a3, 0
        addi $t2, $a1, -1    # Test will be done till NUM - 1

loop1:  div  $a1, $a2        # Test if the current number divide NUM
        mfhi $t3
        beq  $t3, $a3, loop3 # The test has failed for the current number
        addi $a2, $a2, 1
        beq  $a2, $t2, loop2 # The test has been successfull
        j    loop1

loop2:  li   $v0, 4          # Code 4 for service routine print string
        la   $a0, STR2       # address of the string to print
        syscall              # print the string STR2
        li   $v0, 1          # Code 1 for service routine print integer
        lw   $a0, NUM        # address of the Number to print
        syscall              # print the number NUM
        li   $v1, 1          # 1 in v1 for successfull test
        j    EXIT

loop3:  li   $v0, 4          # Code 4 for service routine print string
        la   $a0, STR1       # address of the string to print
        syscall              # print the string STR1
        li   $v0, 1          # Code 1 for service routine print integer
        lw   $a0, NUM        # address of the Number to print
        syscall              # print the number NUM
        li   $v1, 0          # 0 in v1 if test failed
        j    EXIT
EXIT:
```

a) Finden Sie heraus was das oben angegebene Programm berechnet.

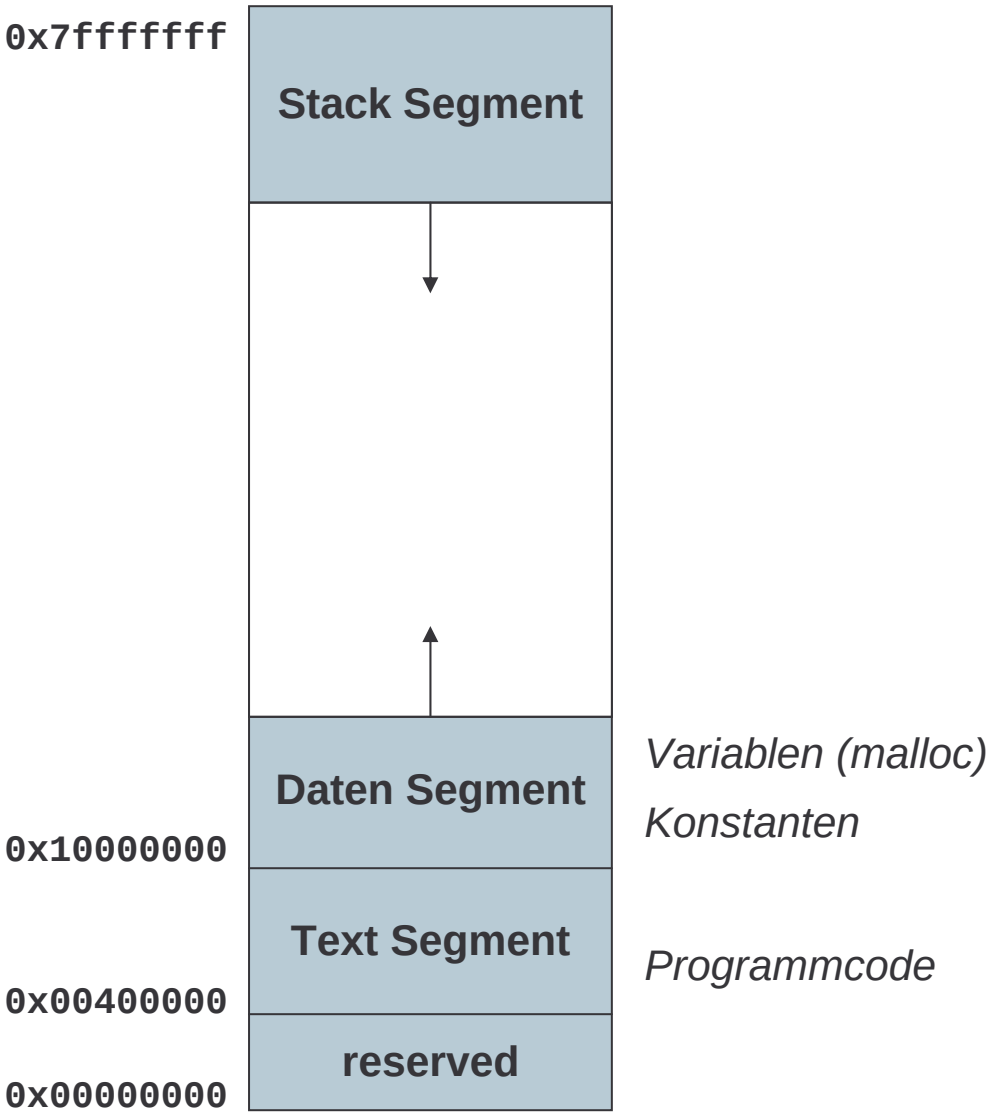
b) Erstellen Sie zu diesem Programm den Programm-Ablaufplan.

Aufgabe 2: Erweiterung

a) Schreiben Sie ein Unterprogramm, das den div-Befehl ersetzt. Das Unterprogramm erhält als Parameter den Wert und den Teiler und liefert den Divisionsrest zurück. Bei dem Funtionsaufruf müssen alle Schritte eines Funtionsaufruf (Parameter Übergabe, Speicherplatzreservierung, Rettung von Register, etc...) deutlich gezeigt werden.

b) Wie muss das obige Assembler-Programm angepasst werden damit diese Unterroutine benutzt werden kann?

Speicheraufteilung



Unterprogramm Aufruf

- **Unterprogrammsprung vorbereiten**

1) Argumente übergeben
   - ersten 4 Argumente → $a0 - $a3
   - restlichen Argumente → Stack

2) Caller-Saved Register sichern
   → nach Aufruf benötigte $t0-$t9 Register

3) aufrufen von **`jal`** (jump and link)
   - atomare Operation:
     sichere PC+4 in $ra
     lade PC mit Unterprogramm Adress

- **Während des Unterprogrammaufrufs**

1) Stack frame aufbauen
   - $sp = $sp – newFrameSize

2) Callee-Saved Register sichern
   - $fp
   - nach Bedarf $ra und $s0 … $s7

3) Setze Frame pointer
   - $fp = $sp + newFrameSize

- **Rückkehr vom Aufruf (Callee)**

0) function→ place result in **$v0**

1) Callee-Saved Register wiederherstellen, die zu Beginn gesichert wurden, inkl. **$fp**

2) Pop Stack Frame
   - addiere newFrameSize zu $sp

3) Zurückspringen an Adresse $ra
   → Behehl **jr**

- **Rückkehr vom Aufruf (Caller)**

1) Caller-Saved Register wiederherstellen: $tx

2) Stack: Pop Argumente 5…n

3) ggf. Ergebnis $v0 sichern