PR RECHNERORGANISATION 621.701 – 621.703 **WS 2007/2008**

Institut für Informationstechnologie (ITEC) **Prangl/Taschwer/Timmerer**

Übungsblatt 2

Ü 2.1 MIPS-Grundlagen

Machen Sie sich mit dem MIPS-Assemblerbefehlssatz anhand der im Zeus zur Verfügung stehenden Unterlagen vertraut.

a.) Erklären Sie, welche und wie viele Register in der MIPS-Architektur zur Verfügung stehen und wozu sie verwendet werden.

b.) Erklären Sie den Unterschied zwischen I-, R-, und J-Befehlen. Warum wird zwischen diesen 3 Typen unterschieden?

c.) Mit welchen Befehlen werden Multiplikation und Division von ganzen Zahlen bewerkstelligt, und wo sind die Ergebnisse zu finden?

Ü 2.2 MIPS-Assembler – Speicherzugriffe

Das folgende MIPS-Programmfragment versucht Datenwörter ausgehend von der Basisadresse ($a0) in den Speicherbereich mit der Basisadresse ($a1) zu kopieren. Ist der Wert eines Datenwortes gleich 0, bricht der Kopiervorgang ab. Im Register $v0 wird die Anzahl der kopierten Datenwörter (exklusive dem Terminierungswort) mitprotokolliert.

```
main: addi  $v0,   $zero,  0      # Initialize count
loop: lw    $t0,   0($a0)         # Read next word from source
      sw    $t1,   0($a1)         # Write it to destination
      addi  $a0,   $a0,  1        # Advance pointer to next source
      addi  $a1,   $a1,  1        # Advance pointer to next destination
      beq   $t0,   $zero, loop    # Loop  if word copied != zero
      jr    $ra
```

Es haben sich mehrere Fehler im Programm eingeschlichen; ein Befehl ist sogar verlorengegangen. Wandeln Sie das Programm in eine fehlerfreie Version um.

Ü 2.3 MIPS-Assembler – Speicherzugriffe

Im folgenden Codefragment ist eine 3x4-Matrix gegeben. Ergänzen Sie den Code, sodass in *tpos* die Transposition von *matrix* gespeichert wird!

```
.data
matrix: .word  1,  2,  3, 4
        .word  5,  6,  7, 8
        .word  9, 10, 11, 12
```

```
                    # Transposition:
tpos:   .word 0, 0, 0    # 1, 5, 9
        .word 0, 0, 0    # 2, 6, 10
        .word 0, 0, 0    # 3, 7, 11
        .word 0, 0, 0    # 4, 8, 12

dimx:   .word 4
dimy:   .word 3

.text
.globl main
main:
# lade Wert von dimx in Register $a0
...
# lade Wert von dimy in Register $a1
…
# rechne Transposition
...
jr      $ra
```

Ü 2.4 MIPS-Assembler – Schleifenprogrammierung, Eingabe/Ausgabe

Schreiben Sie ein SPIM-Programm, welches den folgenden Algorithmus verwirklicht:

Input: *n* und *m*, positive ganze Zahlen, eingelesen von der Tastatur.

1.) Dividiere *m* durch *n*, speichere den Rest der Division in *r*.

2.) Wenn $r = 0$, dann ist der Algorithmus beendet, und das Ergebnis ist *n*.

3.) Wenn $r \neq 0$, dann speichere den Wert von *n* in *m* und den Wert von *r* in *n*.

Gehe zurück zu 1.)

Output: Ausgabe des Ergebnisses am Bildschirm.

Was rechnet der Algorithmus aus? Überprüfen Sie, ob die eingelesenen Parameter positive Zahlen sind. Sonst soll als Ergebnis **-1** ausgegeben werden.

Hinweis: Die Eingabe über die Tastatur und die Ausgabe am Bildschirm werden mit `syscall` Befehlen bewerkstelligt.

Ü 2.5 MIPS-Assembler – Zeichenketten, ASCII

Schreiben Sie ein SPIM-Programm, das alle Großbuchstaben eines gegebenen ASCII-Textes durch Kleinbuchstaben ersetzt, und umgekehrt.

Beispiel: Das ist mein Text, geschrieben am 18.8.2005

wird zu: dAS IST MEIN tEXT, GESCHRIEBEN AM 18.8.2005

.data

 # definiert null-terminierte Zeichenkette (String)

mystring: .asciiz “Das ist mein Text, geschrieben am 18.8.2005”