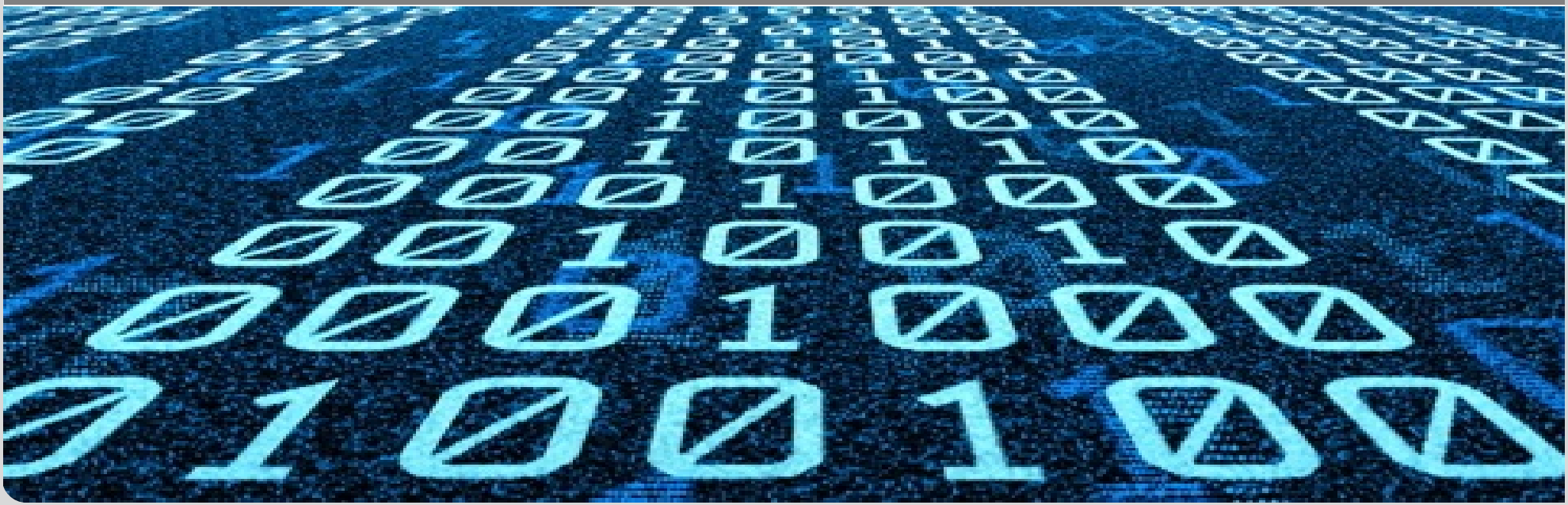
Karlsruher Institut für Technologie

5. Tutorium
MIPS-Assembler

Rechnerorganisation, Tutorium #13

Patrick Röper | 3. Dezember 2019

FAKULTÄT FÜR INFORMATIK



Roadmap

1 Nachbesprechung

2 Aufgaben

Nachbesprechung

- Adressen...
- Assemblerdirektiven != Befehle
- $sX & $txt
- j label & jal label

Roadmap

1 Nachbesprechung

2 Aufgaben

Aufgabe 1.1

Aufgabe

Geben Sie die MIPS-Instruktionen zu den folgenden Pseudoinstruktionen an.

- **b marke**
- **neg $s3, $s2**

Aufgabe 1.1

Aufgabe

Geben Sie die MIPS-Instruktionen zu den folgenden Pseudoinstruktionen an.

- **b marke**
- **neg $s3, $s2**

Lösung

MIPS-Instruktionen:

- **b marke** wird ersetzt durch **bgez $zero, marke**
- **neg $s3, $s2** wird ersetzt durch **sub $s3,$zero,$s2**

Aufgabe 1.2

Aufgabe

Was bewirkt die Assemblerdirektive **.align 3** ?

Aufgabe 1.2

Aufgabe

Was bewirkt die Assemblerdirektive **.align 3** ?

Lösung

Assemblerdirektive **.align 3** :

Bewirkt, dass die folgenden Daten an der nächstmöglichen Adresse gespeichert werden, die durch $2^3 = 8$ teilbar ist.

Aufgabe 1.3

Aufgabe

Warum dürfen bei Arithmetikoperationen mit doppelter Genauigkeit nur die Register mit gerader Registernummer verwendet werden?

Aufgabe 1.3

Aufgabe

Warum dürfen bei Arithmetikoperationen mit doppelter Genauigkeit nur die Register mit gerader Registernummer verwendet werden?

Lösung

Die Register des Prozessors sind 32 bit breit. Bei Gleitkommaoperationen mit doppelter Genauigkeit sind die Operanden 64 Bit breit -> Jeder Operand wird in zwei aufeinander folgenden Register gespeichert.

Aufgabe 1.4

Aufgabe

Welche Gründe machen die Programmierung der MIPS-Architektur schwierig?

Aufgabe 1.4

Aufgabe

Welche Gründe machen die Programmierung der MIPS-Architektur schwierig?

Lösung

- Verzögertes Laden
- Verzögertes Verzweigen
- Eingeschränkter Befehlssatz
- Einschränkungen im Befehlsformat
- Wenige Adressierungsarten

Aufgabe 1.5

Aufgabe

Welche Adressen haben A, B, C und D im folgenden MIPS-Programmabschnitt?

```
        .data 0x10000003
        .align 3
A:      .byte 6, 5
B:      .word 7, 4
C:      .double 3.1415
D:      .float 2.71828
```

Aufgabe 1.5

Lösung

```
        .data 0x10000003
        .align 3
A:      .byte 6, 5
B:      .word 7, 4
C:      .double 3.1415
D:      .float 2.71828
```

Adresse von A : 0x10000008
Adresse von B : 0x1000000C
Adresse von C : 0x10000018
Adresse von D : 0x10000020

Aufgabe 2.1

Aufgabe

Schreiben Sie die folgenden in C gegebenen Kontrollstrukturen in MIPS-Assembler um. Sie dürfen nur die MIPS-Befehle slt, beq und bne verwenden. Zur Speicherung temporärer Variablen verwenden Sie das Register **$at**. Die Variable a ist im Register **$t4**, die Variable b im Register **$s0** abgelegt

Aufgabe 2.1.1

Aufgabe

```
if ( a <= b ) {
    ...
}
marke1 :

if ( a >= b ) {
    ...
}
marke2 :
```

Lösung 2.1.1

Lösung

```
        slt  $at ,$s0 ,$t4
        bne  $at ,$zero , marke1
        ...
marke1 :
        slt  $at ,$t4 ,$s0
        bne  $at ,$zero , marke2
        ...
marke2 :
```

Aufgabe 2.1.2

Aufgabe

```
do {
      marke3 :
      ...
} while ( a != b )
```

Aufgabe 2.1.2

Aufgabe

```
do {
      marke3 :
      ...
} while ( a != b )
```

Lösung

```
marke3 :
          ...
          bne $s0 ,$t4 , marke3
```

Aufgabe 2.2

Aufgabe

Was sind die Unterschiede zwischen einer statischen und einer dynamischen Speicherallokierung?

Lösung 2.2

Lösung

- Statische Speicherallokierung:
  erfolgt während der Assemblierung durch Assemblerdirektiven
  Speicherplatz ist während der gesamten Laufzeit belegt. Eine Veränderung der Größe ist nicht möglich.
- Dynamische Speicherallokierung:
  erfolgt während der Laufzeit durch spezielle Systemaufrufe
  Speicherplatz ist nur bei Bedarf belegt. Es entsteht Aufwand während der Ausführung des Programms um den entsprechenden Speicher zu reservieren und freizugeben

Aufgabe 3

Aufgabe

Setzen Sie die folgenden C Kontrollstrukturen mit MIPS-Assembler um (kein vollstaendiges Programm erforderlich). Die Variablen a, b, i und sum vom Typ int32t sollen hierbei in den Registern **$s0** bis **$s3** abgelegt werden.

Aufgabe 3

Aufgabe

```
1. if (a * 2 < b) {
     a *= 2;
}
2. for (a = 10; a >= 0; a -= 2) {
     b += a;
}
```

Lösung 3

Lösung

```
1.      li $t0 , 2                  # $t0 = 2
        mul $t0 , $s0 , $t0         # $t0 = a * 2
        bge $t0 , $s1 , skip        # $t0 >= b?, dann zu skip
        move $s0 , $t0              # a = a * 2 ($t0 = a * 2)
skip:   ...


2.      li $s0 , 10                 # a = 10
loop:   bltz $s0 , endloop          # a < 0?, dann zu endloop
        add $s1 , $s1 , $s0         # b = b + a
        addi $s0 , $s0 , -2         # a = a - 2
        j loop                      # zu loop
endloop: ...
```

Aufgabe 4.1

Aufgabe

Beschreiben Sie die Funktion der folgenden MIPS-Befehle:

- **addu $t3, $t2, $t1**
- **andi $t3, $t2, 0x2000**
- **slt $t3, $t2, $t1**
- **lui $t3, 0x2000**

Aufgabe 4.1

Lösung

- **addu $t3, $t2, $t1**
  Addition: $t3 = $t2 + $t1 Bereichsüberschreitung wird nicht berücksichtigt.
- **andi $t3, $t2, 0x2000**
  Logisches UND: $t3 =$t2 UND 0x2000
- **slt $t3, $t2, $t1**

```
if ($t2 < $t1)
    $t3 = 1
else
    $t3 = 0
```

- **lui $t3, 0x2000**
  Lade das niederwertigste Halbwort vom Imm (hier 0x2000) in das höchstwertige Halbwort des Regsiters $t3 .

Aufgabe 4.2

Aufgabe

In welchem Register wird die Rücksprungadresse beim Unterprogrammaufruf gesichert?

Aufgabe 4.2

Aufgabe

In welchem Register wird die Rücksprungadresse beim Unterprogrammaufruf gesichert?

Lösung

Register für die Rücksprungadresse: **$ra**

Aufgabe5.1

Aufgabe

Geben Sie für das folgende MIPS-Programmstück nach der Ausführung jedes Befehls den Inhalt des jeweiligen Zielregisters in hexadezimaler Schreibweise an.

```
ori   $s1, $zero, 20
sll   $s2, $s1, 3
slti  $s3, $s2, 100
sub   $s4, $s3, $s2
lui   $s5, -7
```

Aufgabe 5.1

Befehl	Zielregister = (z. B. $s6 = 0x0000 F00A)
`ori $s1, $zero, 20`	`$s1 = 0x0000 0014`
`sll $s2, $s1, 3`	`$s2 = 0x0000 00A0`
`slti $s3, $s2, 100`	`$s3 = 0x0000 0000`
`sub $s4, $s3, $s2`	`$s4 = 0xFFFF FF60`
`lui $s5, -7`	`$s5 = 0xFFF9 0000` (in SPIM: 0xFFFF FFF9)

Aufgabe5.2

Aufgabe

Wie ist die Trennung von Programmen und Daten bei der Ihnen bekannten MIPS-R2000-Architektur realisiert?

Aufgabe5.2

Aufgabe

Wie ist die Trennung von Programmen und Daten bei der Ihnen bekannten MIPS-R2000-Architektur realisiert?

Lösung

Trennung von Programmen und Daten bei der MIPS-R2000-Architektur: Der Hauptspeicher wird in mehrere Segementen unterteilt. Es existiert ein Datensegment für die Daten und ein Textsegment für Programme.

Aufgabe 6

Aufgabe

```
                .data
X:              .word 3
Y:              .word 1 , 3 , 7

    . text
subroutine :    li $v0 , 0
                li $t3 , 0
marke1 :        bge $t3 , $a1 , marke2
                lw $t0 , 0( $a0 )
                mul $t1 , $t0 , $t0
                add $v0 , $v0 , $t1
                addi $a0 , $a0 , 4
                addi , $t3 , $t3 , 1
                b marke1
marke2 :        jr $ra
```

Aufgabe 6

Aufgabe

```
                .globl main

main :          la $a0 , Y
                lw $a1 , x
                jal subroutine

                move $a0 , $v0
                li $v0 , 1
                syscall
                li $v0 , 10
                syscall
                jr $a
```

Aufgabe 6

lösung

Aufruf eines Unterprogramms , welches die Quadratwurzel einer Integerzahl in **$v0** berechnet . Das Ergebnis steht in **$v0**.

Was ihr jetzt kennen und können solltet...

- MIPS Assembler