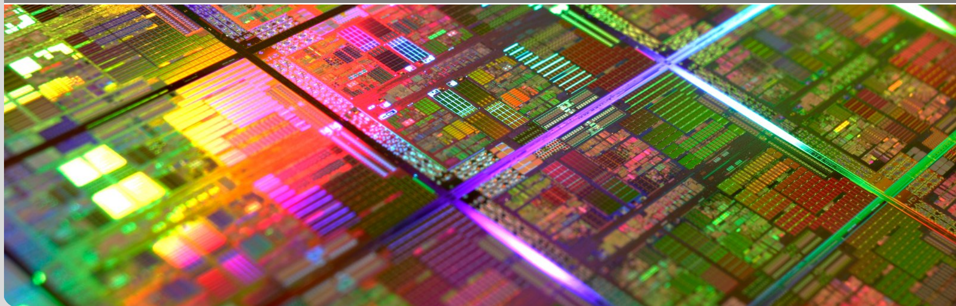
Karlsruher Institut für Technologie

RO-Tutorien 17 und 18

Tutorien zur Vorlesung “Rechnerorganisation”

Christian A. Mandery

TUTORIENWOCHE 7 AM 14.06.2012



Heute

- Einführung in die MIPS-Architektur
- Übungsaufgaben

Was ist MIPS?

Karlsruher Institut für Technologie

- **M**icroprocessor without **I**nterlocked **P**ipeline **S**tages
- Prozessorarchitektur, die ab 1981 in Stanford entwickelt wurde
- Folgt den Designprinzipien einer RISC-Architektur (Reduced Instruction Set Computing)
  - Wenige, elementare Befehle; wenige Befehlsformate; viele Register
  - Festverdrahtetes Steuerwerk statt Mikroprogrammierung
  - Deshalb hohe Taktung und einfaches Pipelining möglich
- Befehlsreferenz: http://ti.itec.uka.de/TI-2/Spim/Befehlssatz.pdf
  - Auf Vorlesungs- und Tutoriumsseite verlinkt
  - Am besten ausdrucken

Was ist MIPS?

Karlsruher Institut für Technologie

- **M**icroprocessor without **I**nterlocked **P**ipeline **S**tages
- Prozessorarchitektur, die ab 1981 in Stanford entwickelt wurde
- Folgt den Designprinzipien einer RISC-Architektur (Reduced Instruction Set Computing)
  - Wenige, elementare Befehle; wenige Befehlsformate; viele Register
  - Festverdrahtetes Steuerwerk statt Mikroprogrammierung
  - Deshalb hohe Taktung und einfaches Pipelining möglich
- Befehlsreferenz: http://ti.itec.uka.de/TI-2/Spim/Befehlssatz.pdf
  - Auf Vorlesungs- und Tutoriumsseite verlinkt
  - Am besten ausdrucken

Was ist MIPS?

- **M**icroprocessor without **I**nterlocked **P**ipeline **S**tages
- Prozessorarchitektur, die ab 1981 in Stanford entwickelt wurde
- Folgt den Designprinzipien einer RISC-Architektur (Reduced Instruction Set Computing)
  - Wenige, elementare Befehle; wenige Befehlsformate; viele Register
  - Festverdrahtetes Steuerwerk statt Mikroprogrammierung
  - Deshalb hohe Taktung und einfaches Pipelining möglich
- Befehlsreferenz: http://ti.itec.uka.de/TI-2/Spim/Befehlssatz.pdf
  - Auf Vorlesungs- und Tutoriumsseite verlinkt
  - Am besten ausdrucken

SPIM

Karlsruher Institut für Technologie

- SPIM := MIPS rückwärts
- Simulator, der eine MIPS32-CPU emuliert und einen Assemblierer enthält
- Verfügbar für Linux, Windows und Mac unter http://spimsimulator.sourceforge.net/

Befehlsformate von MIPS

Karlsruher Institut für Technologie

- Alle Befehlsformate sind 32 Bit breit
- R-Typ ("Register"): 6 Bit Opcode, 3x 5 Bit Register-Nummer, 5 Bit Shift Amount, 6 Bit Funktions-Nummer
- I-Typ ("Immediate"): 6 Bit Opcode, 2x 5 Bit Register-Nummer, 16 Bit Immediate-Wert
- J-Typ ("Jump"): 6 Bit Opcode, 26 Bit Sprungziel

Befehlsformate von MIPS

Karlsruher Institut für Technologie

- Alle Befehlsformate sind 32 Bit breit
- R-Typ (“Register”): 6 Bit Opcode, 3x 5 Bit Register-Nummer, 5 Bit Shift Amount, 6 Bit Funktions-Nummer
- I-Typ (“Immediate”): 6 Bit Opcode, 2x 5 Bit Register-Nummer, 16 Bit Immediate-Wert
- J-Typ (“Jump”): 6 Bit Opcode, 26 Bit Sprungziel

Befehlsformate von MIPS

- Alle Befehlsformate sind 32 Bit breit
- R-Typ ("Register"): 6 Bit Opcode, 3x 5 Bit Register-Nummer, 5 Bit Shift Amount, 6 Bit Funktions-Nummer
- I-Typ ("Immediate"): 6 Bit Opcode, 2x 5 Bit Register-Nummer, 16 Bit Immediate-Wert
- J-Typ ("Jump"): 6 Bit Opcode, 26 Bit Sprungziel

Befehlsformate von MIPS

- Alle Befehlsformate sind 32 Bit breit
- R-Typ ("Register"): 6 Bit Opcode, 3x 5 Bit Register-Nummer, 5 Bit Shift Amount, 6 Bit Funktions-Nummer
- I-Typ ("Immediate"): 6 Bit Opcode, 2x 5 Bit Register-Nummer, 16 Bit Immediate-Wert
- J-Typ ("Jump"): 6 Bit Opcode, 26 Bit Sprungziel

Einige MIPS-Befehle (Arithmetik)

- Grundsätzlich: "`u`" = unsigned, "`i`" = immediate
- `add`, `addu`, `addi`, `addiu`, `sub`, `subu`
- `mult`, `div`, `divu`
- `and`, `andi`, `or`, `ori`, `xor`, `nor`
- `sll`, `srl`, `sra`
- `slt`, `slti`

Einige MIPS-Befehle (Arithmetik)

Karlsruher Institut für Technologie

- Grundsätzlich: “`u`” = unsigned, “`i`” = immediate
- `add`, `addu`, `addi`, `addiu`, `sub`, `subu`
- `mult`, `div`, `divu`
- `and`, `andi`, `or`, `ori`, `xor`, `nor`
- `sll`, `srl`, `sra`
- `slt`, `slti`

Einige MIPS-Befehle (Arithmetik)

Karlsruher Institut für Technologie

- Grundsätzlich: “u” = unsigned, “i” = immediate
- add, addu, addi, addiu, sub, subu
- mult, div, divu
- and, andi, or, ori, xor, nor
- sll, srl, sra
- slt, slti

Einige MIPS-Befehle (Arithmetik)

Karlsruher Institut für Technologie

- Grundsätzlich: “`u`” = unsigned, “`i`” = immediate
- `add`, `addu`, `addi`, `addiu`, `sub`, `subu`
- `mult`, `div`, `divu`
- `and`, `andi`, `or`, `ori`, `xor`, `nor`
- `sll`, `srl`, `sra`
- `slt`, `slti`

Einige MIPS-Befehle (Arithmetik)

Karlsruher Institut für Technologie

- Grundsätzlich: "`u`" = unsigned, "`i`" = immediate
- `add`, `addu`, `addi`, `addiu`, `sub`, `subu`
- `mult`, `div`, `divu`
- `and`, `andi`, `or`, `ori`, `xor`, `nor`
- `sll`, `srl`, `sra`
- `slt`, `slti`

Einige MIPS-Befehle (Arithmetik)

Karlsruher Institut für Technologie

- Grundsätzlich: “`u`” = unsigned, “`i`” = immediate
- `add`, `addu`, `addi`, `addiu`, `sub`, `subu`
- `mult`, `div`, `divu`
- `and`, `andi`, `or`, `ori`, `xor`, `nor`
- `sll`, `srl`, `sra`
- `slt`, `slti`

Einige MIPS-Befehle (Datentransfer)

Karlsruher Institut für Technologie

- `lb`, `lh`, `lw`
- `sb`, `sh`, `sw`
- `mfhi`, `mflo`
- `li` (Pseudobefehl), `lui`
- `mfc0`, `mtc0`

Einige MIPS-Befehle (Datentransfer)

Karlsruher Institut für Technologie

- `lb`, `lh`, `lw`
- `sb`, `sh`, `sw`
- `mfhi`, `mflo`
- `li` (Pseudobefehl), `lui`
- `mfc0`, `mtc0`

Einige MIPS-Befehle (Datentransfer)

Karlsruher Institut für Technologie

- `lb`, `lh`, `lw`
- `sb`, `sh`, `sw`
- `mfhi`, `mflo`
- `li` (Pseudobefehl), `lui`
- `mfc0`, `mtc0`

Einige MIPS-Befehle (Datentransfer)

- `lb`, `lh`, `lw`
- `sb`, `sh`, `sw`
- `mfhi`, `mflo`
- `li` (Pseudobefehl), `lui`
- `mfc0`, `mtc0`

Einige MIPS-Befehle (Datentransfer)

Karlsruher Institut für Technologie

- `lb`, `lh`, `lw`
- `sb`, `sh`, `sw`
- `mfhi`, `mflo`
- `li` (Pseudobefehl), `lui`
- `mfc0`, `mtc0`

Einige MIPS-Befehle (Kontrollfluss)

- `j`, `jr`, `jal`
- `syscall`
- `beq`, `bne`
- `bgt`, `blt`, `bge`, `ble`, `bgtu`, `bgtz` (alles Pseudobefehle)

Einige MIPS-Befehle (Kontrollfluss)

- `j`, `jr`, `jal`
- `syscall`
- `beq`, `bne`
- `bgt`, `blt`, `bge`, `ble`, `bgtu`, `bgtz` (alles Pseudobefehle)

Einige MIPS-Befehle (Kontrollfluss)

Karlsruher Institut für Technologie

- `j`, `jr`, `jal`
- `syscall`
- `beq`, `bne`
- `bgt`, `blt`, `bge`, `ble`, `bgtu`, `bgtz` (alles Pseudobefehle)

Einige MIPS-Befehle (Kontrollfluss)

Karlsruher Institut für Technologie

- `j`, `jr`, `jal`
- `syscall`
- `beq`, `bne`
- `bgt`, `blt`, `bge`, `ble`, `bgtu`, `bgtz` (alles Pseudobefehle)

Pseudobefehle

- Erweiterung des Befehlssatzes, ohne dass sich die Komplexität des CPU-Designs erhöht
- Werden nicht (direkt) auf einen Maschinenbefehl abgebildet, sondern durch einen oder mehrere native Befehle ersetzt
- Anwendungsfälle:
  - Anbieten häufig verwendeter Spezialfälle zur Vereinfachung der Programmierung (z.B. `neg`)
  - Ausgleich von Beschränkungen des Befehlssatzes (z.B. `li`)

Assemblerdirektiven

Karlsruher Institut für Technologie

- Beginnen in MIPS mit einem Punkt
- Werden nicht (direkt) in Maschinencode umgesetzt
- Steuern das Verhalten des Assemblierers
  - Wahl des Segments
  - Reservierung von Speicher
  - Veränderung der Art, wie Daten abgelegt werden (z.B. Alignment)
- Wichtige Assemblerdirektiven: `.data`, `.text`, `.globl`, `.ascii`, `.asciiz`, `.byte`, `.half`, `.word`, `.float`, `.double`, `.space`, `.align`

Assemblerdirektiven

Karlsruher Institut für Technologie

- Beginnen in MIPS mit einem Punkt
- Werden nicht (direkt) in Maschinencode umgesetzt
- Steuern das Verhalten des Assemblierers
  - Wahl des Segments
  - Reservierung von Speicher
  - Veränderung der Art, wie Daten abgelegt werden (z.B. Alignment)
- Wichtige Assemblerdirektiven: `.data`, `.text`, `.globl`, `.ascii`, `.asciiz`, `.byte`, `.half`, `.word`, `.float`, `.double`, `.space`, `.align`

Systemaufrufe

Karlsruher Institut für Technologie

- Werden in MIPS mit dem Befehl `syscall` ausgelöst
- Erlauben die Nutzung von vorgegebenen Betriebssystemfunktionen
  - Details: Vorlesung Betriebssysteme (vormals Systemarchitektur)
- Aufrufkonvention:
  - Nummer des gewünschten Systemaufrufs muss vorher in das Register `$v0` geschrieben werden
  - Übergabe von Parametern und Rückgabewerten ebenfalls über Register

Systemaufrufe

Karlsruher Institut für Technologie

- Werden in MIPS mit dem Befehl `syscall` ausgelöst
- Erlauben die Nutzung von vorgegebenen Betriebssystemfunktionen
  - Details: Vorlesung Betriebssysteme (vormals Systemarchitektur)
- Aufrufkonvention:
  - Nummer des gewünschten Systemaufrufs muss vorher in das Register `$v0` geschrieben werden
  - Übergabe von Parametern und Rückgabewerten ebenfalls über Register

Liste der SPIM-Systemaufrufe

Karlsruher Institut für Technologie

- Ausgabe von Werten:
  - `print_int` (#1), `print_float` (#2), `print_double` (#3), `print_string` (#4)
  - Parameter: Integer in `$a0`, Gleitkommazahl in/ab `$f12`, Startadresse des Strings in `$a0`
- Einlesen von Werten:
  - `read_int` (#5), `read_float` (#6), `read_double` (#7), `read_string` (#8)
  - Parameter: Rückgabe des Integers in `$v0`, Gleitkommazahl in/ab `$f0`, Startadresse des Stringpuffers in `$a0`, maximale Stringlänge in `$a1`
- Sonstiges:
  - `sbrk` (#9): Allokiert Speicherblock der Größe `$a0` Bytes und schreibt die Startadresse in `$v0`
  - `exit` (#10): Beendet die Ausführung

Liste der SPIM-Systemaufrufe

Karlsruher Institut für Technologie

- Ausgabe von Werten:
  - `print_int` (#1), `print_float` (#2), `print_double` (#3), `print_string` (#4)
  - Parameter: Integer in `$a0`, Gleitkommazahl in/ab `$f12`, Startadresse des Strings in `$a0`
- Einlesen von Werten:
  - `read_int` (#5), `read_float` (#6), `read_double` (#7), `read_string` (#8)
  - Parameter: Rückgabe des Integers in `$v0`, Gleitkommazahl in/ab `$f0`, Startadresse des Stringpuffers in `$a0`, maximale Stringlänge in `$a1`
- Sonstiges:
  - `sbrk` (#9): Allokiert Speicherblock der Größe `$a0` Bytes und schreibt die Startadresse in `$v0`
  - `exit` (#10): Beendet die Ausführung

Liste der SPIM-Systemaufrufe

Karlsruher Institut für Technologie

- Ausgabe von Werten:
  - `print_int` (#1), `print_float` (#2), `print_double` (#3), `print_string` (#4)
  - Parameter: Integer in `$a0`, Gleitkommazahl in/ab `$f12`, Startadresse des Strings in `$a0`
- Einlesen von Werten:
  - `read_int` (#5), `read_float` (#6), `read_double` (#7), `read_string` (#8)
  - Parameter: Rückgabe des Integers in `$v0`, Gleitkommazahl in/ab `$f0`, Startadresse des Stringpuffers in `$a0`, maximale Stringlänge in `$a1`
- Sonstiges:
  - `sbrk` (#9): Allokiert Speicherblock der Größe `$a0` Bytes und schreibt die Startadresse in `$v0`
  - `exit` (#10): Beendet die Ausführung

Hello World, MIPS!

Karlsruher Institut für Technologie

```
        .data
hello:  .asciiz "Hello World!\n"

        .text
        .globl main

main:   li $v0, 4
        la $a0, hello
        syscall
        jr $ra
```

Übungsaufgabe 1.1

```
            .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
```

```
# Fortsetzung von links
            .globl main
main:       la $a0, Y
            lw $a1, x
            jal subroutine

            move $a0, $v0
            li $v0, 1
            syscall

            li $v0, 10
            syscall
            jr $ra
```

Welche Funktion das Unterprogramm subroutine? Welche Ausgabe hat das gesamte Programm?

Übungsaufgaben 1.2 - 1.4

Karlsruher Institut für Technologie

2. Geben Sie die MIPS-Instruktionen zu den folgenden Pseudoinstruktionen an:
   - `b marke`
   - `neg $s3, $s2`
3. Was bewirkt die Assemblerdirektive `.align 3`?
4. Warum dürfen bei Arithmetikoperationen mit doppelter Genauigkeit nur die Register mit gerader Registernummer verwendet werden?

Übungsaufgaben 1.2 - 1.4

2. Geben Sie die MIPS-Instruktionen zu den folgenden Pseudoinstruktionen an:
   - `b marke`
   - `neg $s3, $s2`
3. Was bewirkt die Assemblerdirektive `.align 3`?
4. Warum dürfen bei Arithmetikoperationen mit doppelter Genauigkeit nur die Register mit gerader Registernummer verwendet werden?

Übungsaufgaben 1.2 - 1.4

2. Geben Sie die MIPS-Instruktionen zu den folgenden Pseudoinstruktionen an:
   - `b marke`
   - `neg $s3, $s2`
3. Was bewirkt die Assemblerdirektive `.align 3`?
4. Warum dürfen bei Arithmetikoperationen mit doppelter Genauigkeit nur die Register mit gerader Registernummer verwendet werden?

Übungsaufgaben 1.2 - 1.4

2. Geben Sie die MIPS-Instruktionen zu den folgenden Pseudoinstruktionen an:
   - `b marke`
   - `neg $s3, $s2`
3. Was bewirkt die Assemblerdirektive `.align 3`?
4. Warum dürfen bei Arithmetikoperationen mit doppelter Genauigkeit nur die Register mit gerader Registernummer verwendet werden?

Übungsaufgaben 1.5 - 1.6

Karlsruher Institut für Technologie

5. 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
```

6. Welche Gründe machen die Programmierung der MIPS-Architektur schwierig?

Übungsaufgaben 1.5 - 1.6

Karlsruher Institut für Technologie

5. 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
```

6. Welche Gründe machen die Programmierung der MIPS-Architektur schwierig?

Übungsaufgabe 2.1

Schreiben Sie die folgenden 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.

1.
```
if ( a <= b ) { ... }
marke1:
```

2.
```
if ( a >= b ) { ... }
marke2:
```

3.
```
do {
  marke3: ...
          ...
} while ( a != b );
```

Übungsaufgabe 2.1

Schreiben Sie die folgenden 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.

1.
```
if ( a <= b ) { ... }
marke1:
```

2.
```
if ( a >= b ) { ... }
marke2:
```

3.
```
do {
  marke3: ...
          ...
} while ( a != b );
```

Übungsaufgabe 2.1

Schreiben Sie die folgenden 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.

1. ```
   if ( a <= b ) { ... }
   marke1:
   ```

2. ```
   if ( a >= b ) { ... }
   marke2:
   ```

3. ```
   do {
     marke3: ...
             ...
   } while ( a != b );
   ```
   ```

Übungsaufgabe 2.2

Karlsruher Institut für Technologie

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

Übungsaufgabe 3.1

Karlsruher Institut für Technologie

Setzen Sie die folgende C-Kontrollstruktur mit MIPS-Assembler um. Die Variablen `a` und `b` vom Typ `int32_t` sollen hierbei in den Registern `$s0` und `$s1` abgelegt werden.

```
for (a + 2 < b) {
  a += 2;
}
```

Übungsaufgabe 3.2

Setzen Sie die folgende C-Kontrollstruktur mit MIPS-Assembler um. Die Variablen `a` und `b` vom Typ `int32_t` sollen hierbei in den Registern `$s0` und `$s1` abgelegt werden.

```
for (a = 10; a >= 0; a -= 2) {
  b += a;
}
```

Übungsaufgabe 3.3

Karlsruher Institut für Technologie

Setzen Sie die folgende C-Kontrollstruktur mit MIPS-Assembler um. Die Variablen `i` und `sum` vom Typ `int32_t` sollen hierbei in den Registern `$s2` und `$s3` abgelegt werden. Das Array `array` ist im Datensegment durch eine `array: .word ...` Direktive abgelegt.

```
int array[100];
...
sum = 0;
for (i = 0; i < 100; i++)
  sum = sum + array[i];
```

Übungsaufgabe 4

Karlsruher Institut für Technologie

1. Was ändert sich am Assembler-Code der Teilaufgabe 3.3, wenn die Variablen im Datensegment abgelegt und mit Marken entsprechend der Namen der Variablen versehen sind?
2. Beschreiben Sie die Funktion der folgenden MIPS-Befehle:
   - addu $t3, $t2, $t1
   - andi $t3, $t2, 0x2000
   - slt $t3, $t2, $t1
   - lui $t3, 0x2000
3. In welchem Register wird die Rücksprungadresse beim Unterprogrammaufruf gesichert?

Übungsaufgabe 4

Karlsruher Institut für Technologie

1. Was ändert sich am Assembler-Code der Teilaufgabe 3.3, wenn die Variablen im Datensegment abgelegt und mit Marken entsprechend der Namen der Variablen versehen sind?
2. Beschreiben Sie die Funktion der folgenden MIPS-Befehle:
   - addu $t3, $t2, $t1
   - andi $t3, $t2, 0x2000
   - slt $t3, $t2, $t1
   - lui $t3, 0x2000
3. In welchem Register wird die Rücksprungadresse beim Unterprogrammaufruf gesichert?

Übungsaufgabe 4

Karlsruher Institut für Technologie

1. Was ändert sich am Assembler-Code der Teilaufgabe 3.3, wenn die Variablen im Datensegment abgelegt und mit Marken entsprechend der Namen der Variablen versehen sind?
2. Beschreiben Sie die Funktion der folgenden MIPS-Befehle:
   - addu $t3, $t2, $t1
   - andi $t3, $t2, 0x2000
   - slt $t3, $t2, $t1
   - lui $t3, 0x2000
3. In welchem Register wird die Rücksprungadresse beim Unterprogrammaufruf gesichert?

Übungsaufgabe 5

Karlsruher Institut für Technologie

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

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

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

Übungsaufgabe 5

Karlsruher Institut für Technologie

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

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

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

Fertig!

Quelle: http://xkcd.com/749/