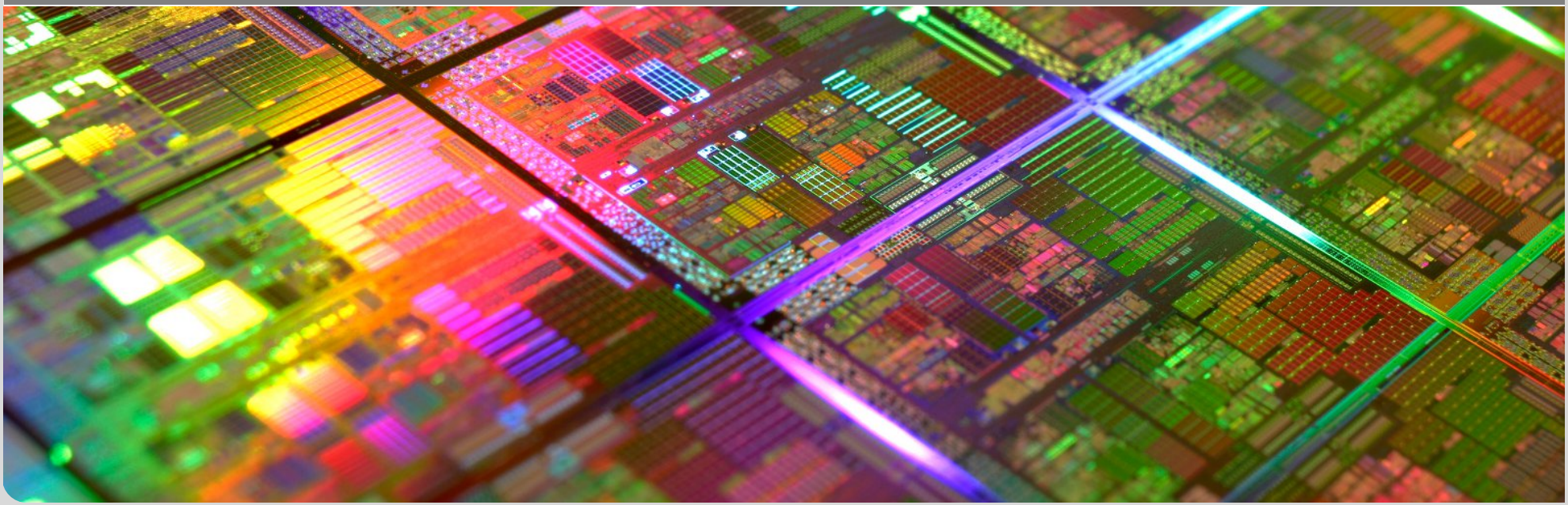
Karlsruher Institut für Technologie

RO-Tutorien 3 / 6 / 12

Tutorien zur Vorlesung “Rechnerorganisation”

Christian A. Mandery

WOCHE 6 AM 03./04.05.2013



Heute

- Einführung in die MIPS-Architektur
- Hello World, MIPS!

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

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

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

- 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

- 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

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)

- 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)

- 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)

- 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)

- 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)

- 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)

- `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)

- `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)

- `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)

- `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)

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

- 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

- 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

- 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

- 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

- 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

- 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

- 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

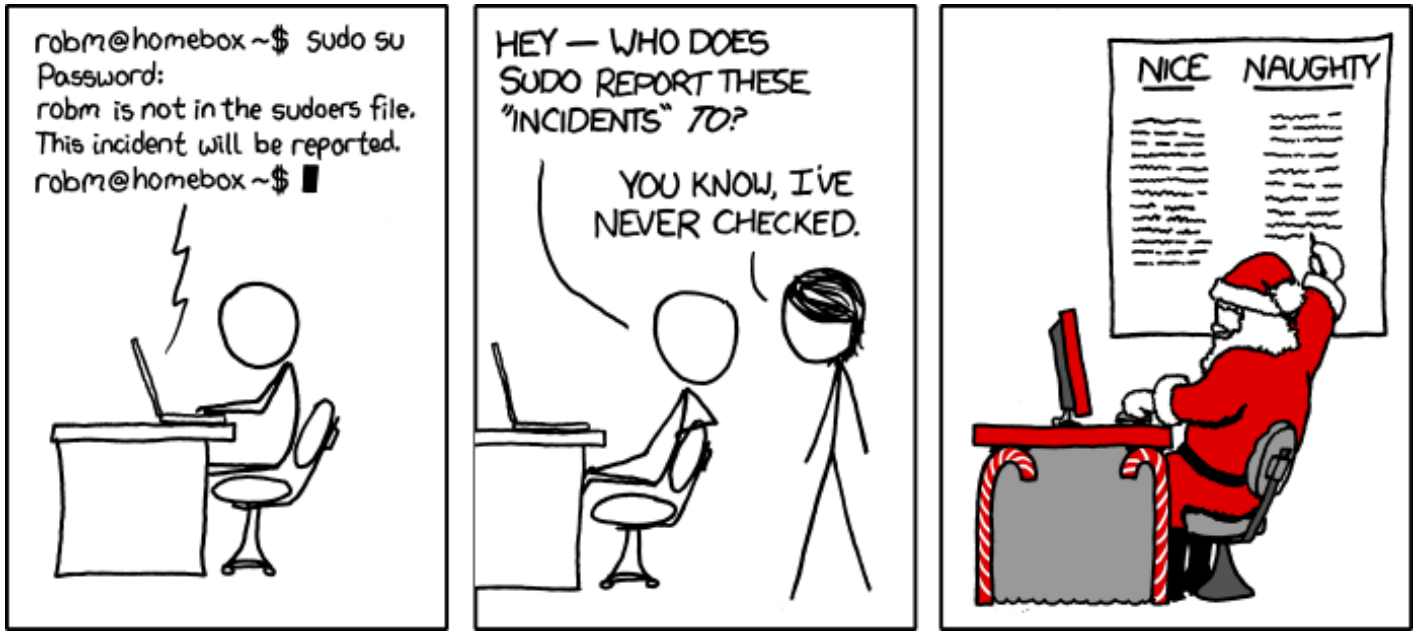
Hello World, MIPS!

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

        .text
        .globl main

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

Fertig!



Quelle: http://xkcd.com/838/