Ostfalia
Hochschule für angewandte Wissenschaften

RISC - Architekturen

Design Digitaler Systeme

Prof. Dr.-Ing. Rainer Bermbach

Übersicht

- CISC - RISC
- Hintergrund
- Merkmale von RISC-Architekturen
- Beispielarchitektur SPARC
- Zusammenfassung

1 CISC - RISC

- CISC
  - Complex Instruction Set Computer
  - „übliche“ Universalprozessoren
  - umfangreicher Befehlssatz
    - Mehrfachoperationen auf wenigen Daten
    - komplexe Datenstrukturen und Adressierungsarten
    - Befehlsformat und Befehlslänge variabel
    - weitgehend mikroprogrammiert

1 CISC - RISC (Forts.)

- RISC
  - Reduced Instruction Set Computer
  - Grundidee:
    Beschränkung des Befehlssatzes auf hauptsächlich benötigte Befehle
  - dadurch „schlankeres“ Design
  - schnellere CPU
  - schnelle Caches ermöglichen andere Ansätze als bisher

2 Hintergrund

- diverse Studien (Anfang der 80er Jahre)
  - Compiler nutzen nicht die Möglichkeiten der Prozessoren
  - deshalb nicht die erwarteten Performance-Gewinne durch komplexe Befehlssätze
  - nur wenige Befehle tatsächlich genutzt:
    - 10 Befehle = 80 % des Programmcodes
    - 21 Befehle = 95 % des Programmcodes
    - 30 Befehle = 99 % des Programmcodes
    - von Befehlssatz mit ca. 200 Befehlen (IBM 370)

2 Hintergrund (Forts.)

- erste Implementierungen bestätigten die Schlussfolgerung:
  - Beschränkung führt zu Performance-Steigerung
- Forschungsprojekte
  - IBM (1979): IBM 801 (ECL, 13 MIPS), Basis der späteren IBM-RISC-Architekturen
  - Uni Berkeley (1982, Patterson, Sequin), Basis der SPARC-Architektur (SUN)
  - Uni Stanford (1981, Hennessy), Stanford MIPS, Basis der Architektur der MIPS-CPU

3 Merkmale von RISC-Architekturen

- deutlich weniger Maschinenbefehle
  - nur wenige „atomare" Befehle
  - alle anderen Funktionen per Software
  - d.h. Verlagerung von Komplexität in den Compiler
  - Merkmal nicht durchgängig anzutreffen:
    - mittlerweile auch RISC mit 100 - 150 Befehlen
    - aber erste SPARC hatte keinen MUL-Befehl
    - aus Statistik begründet, praktisch Fehlschlag:
      - „Statistiker ertrinkt in Bucht mit im Mittel 1m Tiefe"
    - aufgrund der heutigen Möglichkeiten aufgeweicht

3 Merkmale von RISC-Architekturen (Forts.)

- Befehlsformate und Adressierungsmodi eingeschränkt
  - Konzentration auf wirklich erforderliche und einfach zu realisierende Formate/Modi
  - typ. alle Befehle mit gleicher Länge:
    - Prozessorwortbreite
  - gleiches Befehlsformat
    - einheitliche Bedeutung der Bitfelder
  - d.h. geringerer Dekoderaufwand für Befehle

3 Merkmale von RISC-Architekturen (Forts.)

- Befehlsformate/Adressierungsmodi (Forts.)
  - Beispiel Adressberechnung:
    - streng: nur register-indirekt
    - meist: zumindest zusätzliches Displacement
    - auch: Adresse = Addition zweier Registerwerte – kann ohne MMU auskommen
  - Beispiel SPARC:
    - nur 13-Bit-Immediate-Konstante, obere Bits 0
    - alternativ SETHI: oberen 22 Bits, Rest 0
    - ist i.O., da Konstanten meist klein

3 Merkmale von RISC-Architekturen (Forts.)

- Load/Store-Architektur
  - Hauptspeicherzugriffe nur über die Befehle LOAD bzw. STORE
    - alle anderen Operationen arbeiten mit Registeroperanden
  - massiver Einsatz von Pipelining
  - Codegröße wächst
    - einfachere Ablaufsteuerung, Pipeline effektiver
  - praktisch auch Ausnahmen
  - mittlerweile auch LOAD/STORE für verschiedene Datenformate

3 Merkmale von RISC-Architekturen (Forts.)

- starke Registerorientierung
  - viele On-Chip-Register
    - schnellerer Zugriff
    - notwendig wegen Load/Store-Architektur
    - mindestens 32 Register, häufig > 100
  - unterschiedliche Architekturen
    - statische Registerbank
    - Umschaltung von Registerbänken
    - Register Window

3 Merkmale von RISC-Architekturen (Forts.)

- starke Registerorientierung (Forts.)
  - kein klassischer Stack, Parameterübergabe an Unterprogramme über Register (-ausschnitt)
  - Dummy Register 0 (*g0*)
    - liefert beim Lesen immer 0
    - „schreibbar" ohne Speicherung = Mülleimer
      - nur Setzen von Flags
    - Anwendung als 2. Quelloperand bei 3-Adress-Maschinen
      - *cmp r1, r2* ersetzen durch *sub r1, r2, g0*

3 Merkmale von RISC-Architekturen (Forts.)

- keine Mikroprogrammierung
  - durch wenige, einfache Befehle Verzicht auf Mikroprogramm möglich
  - Steuerwerk in Hardware ausgeführt
  - meist schneller in der Ausführung
  - mittlerweile auch besser beherrschbar als zu Anfangszeiten von CISC

3 Merkmale von RISC-Architekturen (Forts.)

- Befehlsausführung in einem Zyklus
  - Gestaltung des Designs durch
    - festverdrahtetes Steuerwerk
    - optimierte Befehlsformate
    - Load/Store-Architektur
  - so, dass möglichst alle (internen) Befehle in einem Maschinenzyklus ausgeführt werden .
  - Streben nach möglichst niedrigem CPI-Wert
    - CPI - clocks per instruction
    - Ideal CPI = 1, heute CPI < 1 (parallele Einheiten)
    - Probleme: s. *Pipelining*

3 Merkmale von RISC-Architekturen (Forts.)

- Befehlsausführung in einem Zyklus
  - hohe Anforderungen an Speicherinterface
    - ohne Cache kaum vernünftige Performance
  - Trennung von Befehlen und Daten
    - Harvard-Architektur
  - niedriger CPI-Wert nur mit Pipelining zu erzielen
  - Hasards müssen aufgelöst werden (in HW oder SW)

3 Merkmale von RISC-Architekturen (Forts.)

- Zusammenspiel mit dem Compiler
  - Verlagerung von Steuerungsaufgaben in den Compiler
    - Registeroptimierung
    - statisches Befehlsscheduling
  - frühe RISC (MIPS) erwarten konfliktfreie Befehlsfolgen (keine Datenhasards)
    - MIPS - Microprocessor without Interlocked Pipeline Stages
  - klarere Struktur erlaubt besser optimierende Compiler

3 Merkmale von RISC-Architekturen (Forts.)

- Zusammenfassung RISC-Merkmale
  - nicht immer alle Merkmale vorhanden bzw. stark ausgeprägt
  - eher Orientierungshilfe zur Klassifizierung
    - Beispiel i486 = klassischer CISC, aber Intel sagt RISC, da Teile des Steuerwerks in Hardware
  - durch Fortschritte in der IC-Technik Unterschiede zwischen CISC & RISC verwischt
    - RISC-Eigenschaften heute allgemein üblich
    - RISC-Prozessor auch komplexer geworden
    - Ausnahme: Embedded Prozessoren

4 Beispielarchitektur SPARC

- Basis: Universität Berkeley
  - Scalable Processor Architecture
- von SUN weiterentwickelt
  - „SUN Processor Architecture“
- erste SPARC CPU sehr klein:
  - 20.000 Gatter (vgl. 100.000 - 200.000 Gatter bei „normalen Prozessoren damals)
- SUN legte Architektur völlig offen
  - um Abhängigkeiten zu vermeiden (fabless)
  - De-Facto-Standard bei Workstations
  - viele Chip- und Workstation-Hersteller

4 Beispielarchitektur SPARC (Forts.)

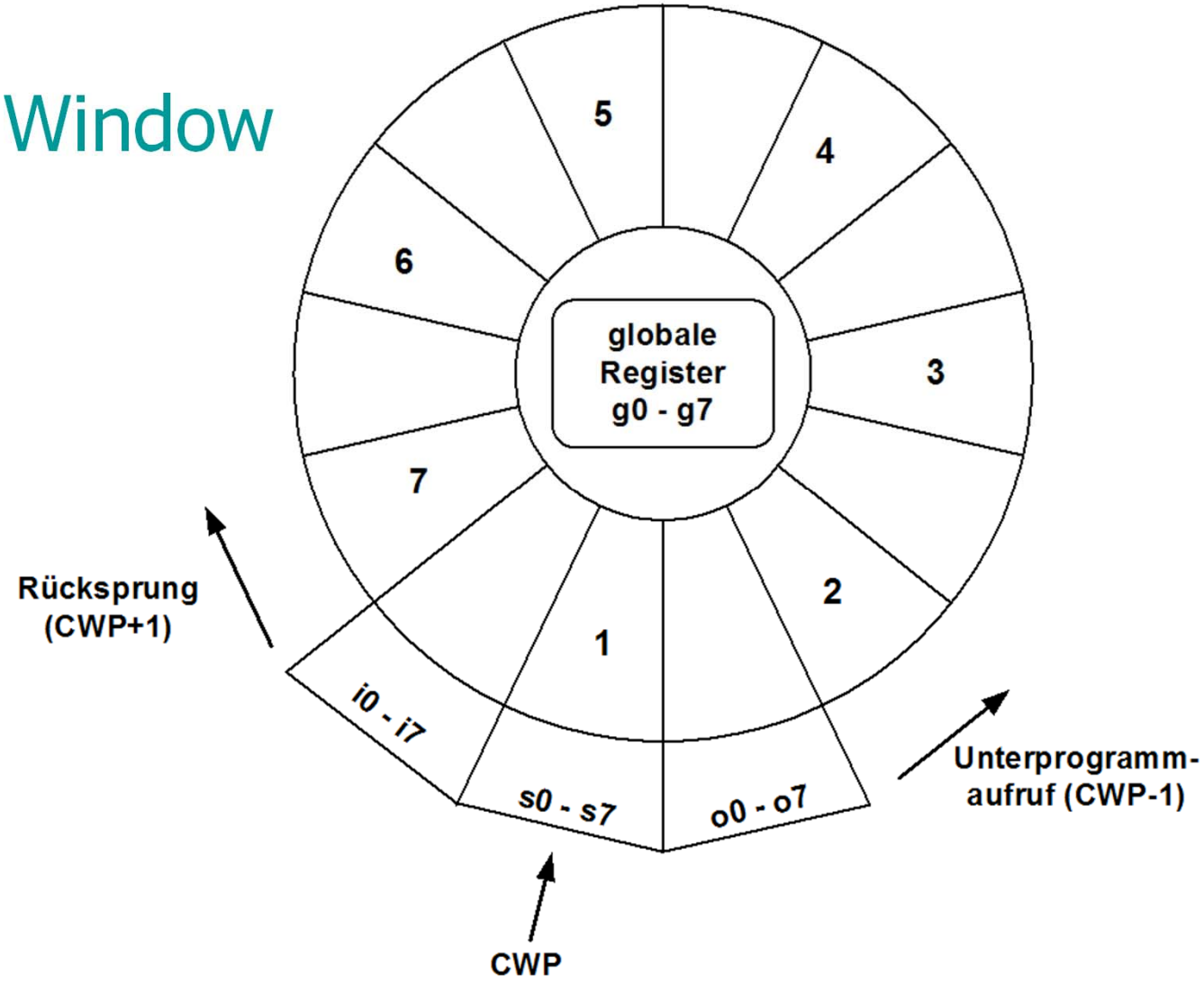
- typ. RISC-Strukturmerkmale (s. o.)
- Besonderheit *Register Windowing*
  - pro Programm/Task stehen 32 Register (32 Bit) zur Verfügung, Zeiger CWP
  - 8 globale Register g0 - g7
  - 24 Register im sog. Window:
    - erste 8 Register: Inputs für Task, i0 - i7
    - mittlere 8 Reg.: lokale Register, s0 - s7
    - letzten 8 Reg.: Outputs der Task, o0 - o7

4 Beispielarchitektur SPARC (Forts.)

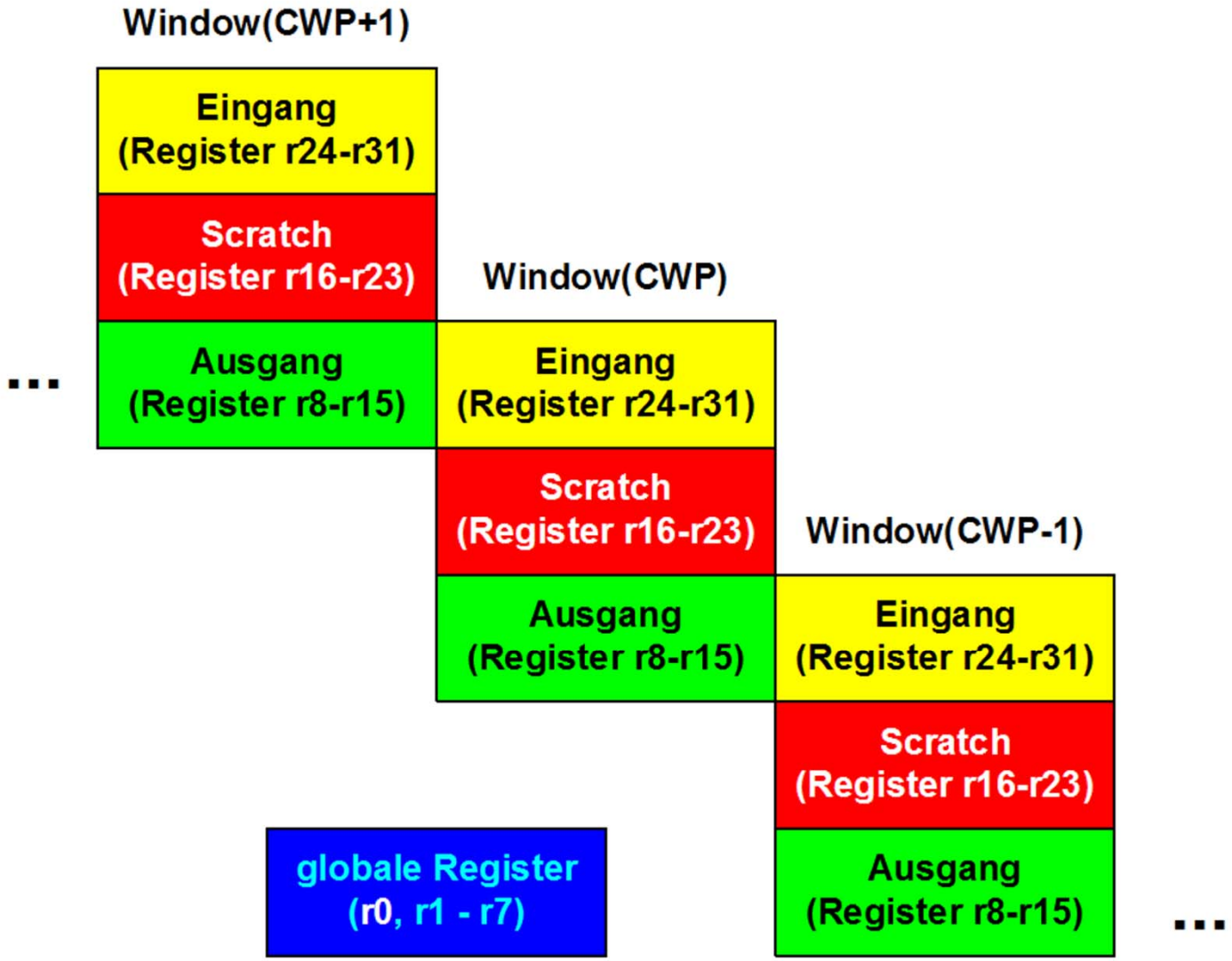
- Register Windowing:
  - SAVE-Befehl = dekrementiere CWP (Current Working Pointer)
  - Registersatz verschiebt sich
    - Ausgangsregister der rufenden Funktion werden Eingangsregister der gerufenen Funktion
    - neuer Satz lokale Register
  - schneller Kontext-Switch
    - nur ein Pointer wird verändert, kein Stack
    - Rücksprungadresse in o7 = i7 der gerufenen Funktion

4 Beispielarchitektur SPARC (Forts.)

- Register Window



4 Beispielarchitektur SPARC (Forts.)



4 Beispielarchitektur SPARC (Forts.)

- Betriebssystem sorgt dafür, dass immer noch ein unbenutztes Register Window zur Verfügung steht
  - damit auch Interrupt-Umschaltung ganz einfach
- Grenze des Register Windowing
  - letztes physikalisches Window benutzt
  - => Trap und auslagern von Windows auf Platte
  - entsprechend laden bei Unterprogrammrückkehr
- für Anwender transparent
  - verschiedene Implementierungen mit unterschiedlicher Speichergröße möglich
    => „Scalable“ Processor Architecture

5 Zusammenfassung

- Unterscheidung CISC - RISC
- Hintergrund:
  - Ausnutzung der Befehlssätze
- Merkmale RISC:
  - weniger Maschinenbefehle
  - beschränkte Befehlsformate/Adressiermodi
  - Load/Store-Architektur und Registerorientierung
  - kein Mikroprogramm (Hardware-Steuerwerk)
  - Befehlsausführung in möglichst einem Zyklus
  - Komplexitätsverlagerung auf den Compiler
- Beispiel SPARC – Register Windowing

Vielen Dank für Ihre Aufmerksamkeit !

ENDE