Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. Dr. h. c. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Übung 2

Dipl.-Inform. Leonard Masing
Dr.-Ing. Oliver Sander

Institut für Technik der Informationsverarbeitung (ITIV)

Hardware/Software Co-Design

Agenda

- Wiederholung ausgewählter Themen
  - RISC vs CISC
  - Pipelining
- Gruppenarbeit
- Vorstellung der Lösung

2.3.1.3 GPP-Arch: RISC vs CISC

- Reduced instruction set computer
  - Schwerpunkt auf Software
  - Nur ein Taktzyklus pro reduzierter Instruktion
  - Load-Store-Architektur
    - Load-Store sind unabhängige Instruktionen
  - Große Codegrößen
  - Benötigt mehr Transistoren für Speicherregister
  - Beispiel: c := a + b
    - LOAD R1, a
    - LOAD R2, b
    - ADD R3, R2, R1
    - STORE c, R3
- Complex instruction set computer
  - Schwerpunkt auf Hardware
  - Mehrere Taktzyklen pro komplexer Instruktion
  - Memory-to-Memory-Architektur
    - Load/Store sind eingebaute Instruktionen
  - Kleine Codegrößen
  - Benötigt mehr Transistoren zum Speichern komplexer Instruktionen
  - Beispiel: c := a + b
    - MOVE R1, a
    - ADD R1, b
    - MOVE c, R1

http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/risc/risccisc/

2.3.2.1 GPP: Pipelining (I)

- **Pipelining** — „Fließband-Bearbeitung“

- „Pipelines beschleunigen die Ausführungsgeschwindigkeit eines Rechners in gleicher Weise wie Henry Ford die Autoproduktion mit der Einführung des Fließbandes revolutionierte.“
  (Peter Wayner 1992)

- Befehlspipeline eines Rechners:
  Die Befehlsbearbeitung wird in n funktionelle Einheiten gegliedert.
  Ausführung geschieht zeitlich überlappend.

2.3.2.1 Phasen der Befehlsausführung einer 5-stufigen Befehlspipeline (I)

- Befehlsbereitstellungsphase (Instruction Fetch):
  - Der Befehl, der durch den Befehlszähler adressiert ist, wird aus dem Hauptspeicher oder dem Cache-Speicher in einen Befehlspuffer geladen. Der Befehlszähler wird weitergeschaltet

- Dekodier- und Operandenbereitstellungsphase (Decode/Operand fetch):
  - Aus dem Operationscode des Maschinenbefehls werden prozessorinterne Steuersignale erzeugt (1. Takthälfte).
    Die Operanden werden aus Registern bereitgestellt (2. Takthälfte)

2.3.2.1 Phasen der Befehlsausführung einer 5-stufigen Befehlspipeline (II)

- Ausführungsphase (ALU Operation):
  - Die Operation wird auf den Operanden ausgeführt.
    Bei Lade-/Speicherbefehlen wird die effektive Adresse berechnet

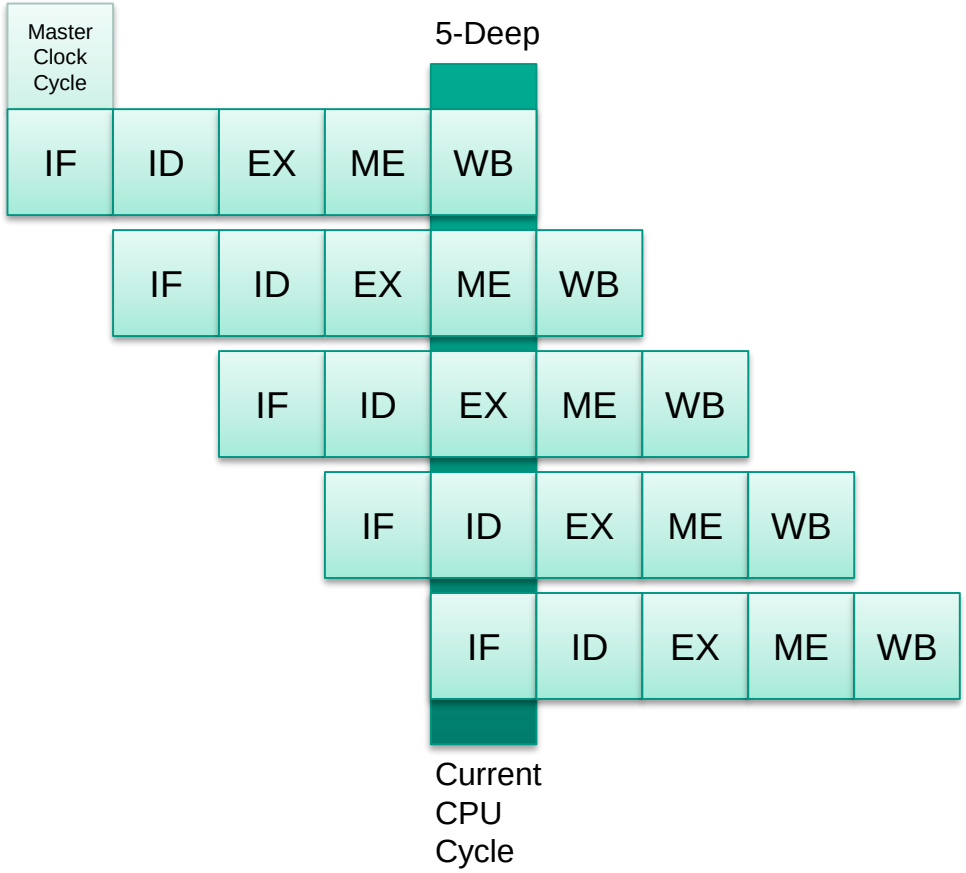
- Speicherzugriffsphase (memory access):
  - Der Speicherzugriff[1] wird durchgeführt

- Resultatspeicherphase (Write Back):
  - Das Ergebnis wird in ein Register geschrieben (1. Takthälfte)

1) Zugriff auf Cache oder RAM, nicht aufs Register!

2.3.2.1 Phasen der Befehlsausführung einer 5-stufigen Befehlspipeline (III)

- IF: Instruction Fetch
- ID: Instruction Decode
- EX: Execute
- MEM: Memory Access
- WB: Write Back



2.3.2.1 Pipeline Konflikte (I)

- Strukturkonflikte
  - Ergeben sich aus Ressourcenkonflikten
    - Die Hardware kann nicht alle mögliche Kombinationen von Befehlen unterstützen, die sich in der Pipeline befinden können
  - Beispiel: Gleichzeitiger Schreibzugriff zweier Befehle auf eine Registerdatei mit nur einem Schreibeingang

- Datenkonflikte
  - Ergeben sich aus Datenabhängigkeiten zwischen Befehlen im Programm
  - Instruktion benötigt das Ergebnis einer vorangehenden und noch nicht abgeschlossenen Instruktion in der Pipeline
  - D.h. ein Operand ist noch nicht verfügbar
    z.B. Echte Datenabhängigkeit, Namensabhängigkeiten

- Steuerkonflikte
  - Treten bei Verzweigungsbefehlen und anderen Instruktionen auf, die den Befehlszähler verändern

2.3.2.1 Auflösung von Pipeline Konflikten

- Softwarebasierend:
  - Aufgabe des Compilers:
    - Erkennen von Datenkonflikten
    - Einfügen von Leeroperationen nach jedem Befehl, der einen Konflikt verursacht
  - Statische Verfahren:
    - Instruction Scheduling, Pipeline Scheduling
    - Eliminieren von Leeroperationen
    - Umordnen der Befehle des Programms (Code-Optimierung)

- Hardware-Lösungen (Dynamische Verfahren)
  - Erkennen von Konflikten
    - Entsprechende Konflikterkennungslogik notwendig!
    - Techniken
      - Leerlauf der Pipeline (Stalling)
      - Forwarding

Arbeitsphase

- Aufgabe 2.01: RISC/CISC
  - Diskussion der Vor- und Nachteile von RISC bzw. CISC
- Aufgabe 2.02: RISC/CISC
  - Assemblercode für RISC bzw. CISC Prozessoren
- Aufgabe 2.03: Pipelining
  - Am Beispiel der DLX Pipeline
- Aufgabe 2.04: Pipelining
  - Programmausführung in der DLX Pipeline

Aufgabe 2.01: RISC/CISC

- Diskutieren Sie die Vor- und Nachteile von RISC bzw. CISC bezüglich
  a) Codegröße
  b) Pipelining
  c) Steuerwerk
  d) Compilerunterstützung

Lösung Aufgabe 2.01a: RISC/CISC - Codegröße

- CISC ++
  - Viele komplex kodierte Befehle unterschiedlicher Länge. Häufiger verwendete Befehle werden kürzer kodiert als komplexe längere Befehle. Es werden relativ wenige Befehle gebraucht um den Code abzuarbeiten und die Codegröße wird geringer.

- RISC --
  - Wenige heterogene gleich lange Befehle (häufig 32-Bit breit). Es werden mehr „einfache“ Befehle gebraucht um den gleichen Code abzuarbeiten. Dadurch müssen häufiger Werte in Registern abgelegt werden. Als Ausgleich ist der Registersatz tendenziell größer und es werden 3-Addressbefehle verwendet, die mehr Bits in der Kodierung verwenden.

- RISC +
  - Durch die Verwendung von komprimierten Befehlen (z.B. 16-Bit ARM Thumb Code) kann die Codegröße auch bei RISC verkleinert werden. Dies geht allerdings zu Lasten der Performanz.

Lösung Aufgabe 2.01b: RISC/CISC - Pipelining

- RISC ++
  - Alle RISC Befehle haben eine vergleichbare Komplexität und können daher gut in einer Pipeline umgesetzt werden, die eine hohe Auslastung der Ressourcen gewährleistet. Es muss nur in einer Stufe ein Speicherzugriff durchgeführt werden.

- CISC --
  - Die komplexen CISC Befehle können nicht in einer Pipeline umgesetzt werden, die eine gute Auslastung der Ressourcen gewährleistet. Die Pipeline muss für komplexe Befehle ausgelegt sein und ist bei einfachen Befehlen unausgelastet. Ein Speicherzugriff kann sowohl beim Laden und Zurückschreiben der Operanden stattfinden.

- CISC +
  - In heutigen Prozessoren (bei Intel ab Pentium Prozessor) werden die CISC Befehle intern in sog. µ-Ops umgesetzt, die dann von einem RISC Kern ausgeführt werden.

Lösung Aufgabe 2.01c: RISC/CISC - Steuerwerk

- RISC ++
  - Durch den regelmäßigen Befehlssatz mit wenigen gleich langen Befehlen ist die Befehlsdekodierung einfacher, was Zeit und Fläche spart.

- CISC --
  - Durch die vielen komplexen Befehle mit unterschiedlicher Länge ist die Befehlskodierung sehr aufwändig. Alleine die Einheit zur Befehlslängenerkennung in heutigen X86-Prozessoren hat die Größe eines PowerPCs.

Lösung Aufgabe 2.01d: RISC/CISC - Compilerunterstützung

- RISC +
  - Für Compiler ist der Umgang mit dem großen homogenen Registersatz einfacher und es kann vergleichsweise guter Code erzeugt werden.

- CISC -
  - CISC Prozessoren sind für den Menschen einfacher in Assembler zu programmieren, stellen aber den Compiler vor Herausforderungen. Die wenigen Spezialregister wirken sich negativ auf die Codeerzeugung aus. So wirkt es sich nachteilig aus, wenn die Befehlsselektion und Registerallokation unabhängig voneinander betrachtet werden.

- Vor- und Nachteile von RISC bzw. CISC:
  - Codegröße
  - Pipelining
  - Steuerwerk
  - Compilerunterstützung
  - …

Aufgabe 2.02: RISC/CISC

- Gegeben ist folgender Assemblercode:

```
a:      word 10
b:      word 20
c:      word

_xyz:
        mov r0, [a]
        add r0, [b]
        mul r0, [b]
        mov [c], r0
```

a) Handelt es sich bei dem Prozessor, der den Assemblercode ausführen kann, um eine RISC oder CISC Maschine? Begründen Sie Ihre Antwort.
b) Was macht der Code?
c) Portieren Sie den Code auf den jeweils anderen Maschinentypen.

Lösung Aufgabe 2.02a,b: RISC/CISC

a) Handelt es sich bei dem Prozessor, der den Assemblercode ausführen kann, um eine RISC oder CISC Maschine? Begründen Sie Ihre Antwort.

- Bei dem Code handelt es sich um eine CISC Maschine. Es gibt keine getrennten Befehle für den Speicherzugriffe (Load-Store Befehle), sondern der Speicherzugriffe wird innerhalb eines Transferbefehl oder arithmetisch-logischen Befehl durchgeführt.

a) Was macht der Code?

- Der Quellcode berechnet c = (a+b)*b, wobei die Werte a und b aus dem Speicher geladen und das Ergebnis im Speicher abgelegt wird.

Lösung Aufgabe 2.02c: RISC/CISC

c) Portieren Sie den Code auf den jeweils anderen Maschinentypen.

CISC:

```
a:      word 10
b:      word 20
c:      word

_xyz:
        mov r0, [a]
        add r0, [b]

        mul r0, [b]
        mov [c], r0
```

RISC:

```
a:      word 10
b:      word 20
c:      word

_xyz:
        load r0, [a]
        load r1, [b]
        add r0, r0, r1
        mul r0, r0, r1
        store [c], r0
```

- Assemblercode:
  - Für RISC oder CISC Maschinen.
  - Was macht der Code?
  - Portieren von Code auf anderen Maschinentypen.

Aufgabe 2.03: Pipelining

a) Handelt es sich um eine RISC oder CISC Pipeline?

b) Erklären Sie den Unterschied zwischen der WB und MEM Stufe?

c) Wie viele Takte muss ein Befehl innerhalb der Pipeline angehalten werden, der das Ergebnis des vorherigen Befehls als Eingabe verwendet?

d) In welcher Pipelinestufe würde dieser Befehl angehalten werden, wenn die Pipeline über eine automatisch Konfliktbehandlung verfügt?

Lösung Aufgabe 2.03: Pipelining

a) Handelt es sich um eine RISC oder CISC Pipeline?

- Es handelt sich um eine RISC Pipeline.

a) Erklären Sie den Unterschied zwischen der WB und MEM Stufe?

- In der WB-Phase wird ein Schreibzugriff auf das Registerfile durchgeführt, also das Ergebnis ins Register geschrieben. In der MEM Phase wird der Datenzugriff (Lese oder Schreiben) auf den Hauptspeicher durchgeführt.

b) Wie viele Takte muss ein Befehl innerhalb der Pipeline angehalten werden, der das Ergebnis des vorherigen Befehls als Eingabe verwendet?

- Der Befehl muss 2 Zyklen warten, wenn ID den Registerinhalt in der 2ten Takthälfte liest und WB bereits in der 1ten Takthälfte schreibt.

a) In welcher Pipelinestufe würde dieser Befehl angehalten werden, wenn die Pipeline über eine automatische Konfliktbehandlung verfügt?

- Innerhalb der ID-Phase.

- Pipelining:
  - RISC oder CISC
  - Unterschied zwischen Stufen
    - Abgrenzung MEM und WB
  - Pipeline angehalten bei Konflikten
    - Konfliktbehandlung

Aufgabe 2.04: Pipelining

a) Markieren Sie alle Datenabhängigkeiten im ASM Code.

b) Skizzieren Sie den Ablauf der Pipeline. Markieren Sie evtl. auftretende Pausen. Für die Zugriffe auf Variablen können Sie davon ausgehen, dass diese über einen Cache rechtzeitig zur Verfügung stehen. Die Register werden in der Write-Back Phase in der ersten Takthälfte geschrieben und in der ID Phase in der zweiten Takthälfte geladen. Die Pipeline verfügt über einen automatischen Stall-Mechanismus, der die Ausführung so lange anhält bis der Konflikt aufgelöst ist.

c) Gehen Sie nun davon aus, dass die Pipeline über keine automatische Konflikterkennung hat. Füllen Sie den ASM-Code mit NOP Befehlen auf, so dass das Programm korrekt ausgeführt wird.

d) Gehen Sie nun davon aus, dass die Pipeline über eine Forwarding-Technik verfügt, so dass ein berechneter oder geladener Wert direkt an die EX-Phase weitergeleitet werden kann. Markieren Sie in der Lösung von Teilaufgabe c) sämtliche NOPs, die damit eingespart werden können.

Aufgabe 2.04: Pipelining

- Auf einem Prozessor mit einer 5-stufiger DLX Pipeline wird das folgende Programm ausgeführt:

```
(1) load R2,[Var1]      ;Lade den Inhalt von Var1
(2) load R1,[Var2]      ;Lade den Inhalt von Var2
(3) add R1,R2           ;R1 = R1 + R2
(4) load R3,[Var3]      ;Lade den Inhalt von Var3
(5) load R4,[Var4]      ;Lade den Inhalt von Var4
(6) sub R4,R3           ;R4 = R4 - R3
(7) add R1,R4           ;R1 = R1 + R4
(8) store [Res],R1      ;Speichere Resultat nach Res
```

- Pipelining in 5-stufiger DLX Pipeline
  - Datenabhängigkeiten
  - Ablauf der Pipeline
    - automatischer Stall-Mechanismus
  - keine automatische Konflikterkennung
    - NOP-Befehle
  - Forwarding-Technik