Ausführungszeit $t[s] = \frac{\text{Taktzyklen [Takte]}}{\text{Frequenz [Hz]}} = \frac{C}{f}$

Leistung absolut $L_{abs}[MIPS] = \frac{\text{Befehlsanzahl}}{\text{Ausführungszeit [s]} * 10^6} = \frac{n}{t * 10^6}$

Leistung relativ $L_{rel}[MIPS] = \frac{\text{Referenzzeit [s]}}{\text{Ausführungszeit [s]}} * \text{RefLeistung [MIPS]} = \frac{t_{ref}}{t_{mess}} * L_{ref}$

Clocks per Instruction $CPI = \frac{\text{Taktzyklen [Takte]}}{\text{Befehlsanzahl}} = \frac{C}{n}$

Gewichtete mittlere $CPI_G = \sum(CPI_{Bef.gr.} * \text{Rel.Häufigkeit}_{Bef.gr.}) = \sum_{i=1}^{n}(CPI_i * p_i)$

Instructions per Clock $IPC = \frac{Befehlsanzahl}{\text{Taktzyklen [Takt]}} = \frac{n}{C}$

Speedup $S_n = \frac{1}{AnteilSeriell + Overhead + \frac{AnteilParallel}{AnzahlProzessoren}} = \frac{1}{A_{seriell} + o(n) + \frac{A_{parallel}}{n}}$

Effizienz $E_n = \frac{Speedup}{AnzahlProzessoren} = \frac{S_n}{n}$

Bezeichnung	Konflikterkennung	Issue-Struktur	Scheduling	Hauptmerkmal	Beispiele
Superskalar	Hardware	Dynamisch	Statisch	In-order Execution	Sun UltraSPARC II/ III
Out of Order	Hardware	Dynamisch	Dynamisch mit Spekulation	Out of Order mit Spekulation	Pentium III, Pentium 4, MIPS 10000
VLIW	Software	Statisch	Statisch	Keine Konflikte	Trimedia, diverse DSPs

Prozessorarchitekturen

CISC complex instruction set computer

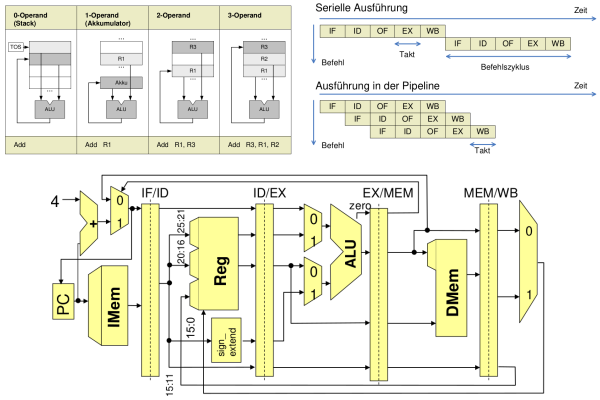
- Einfache und komplexe Befehle
- Heterogener Befehlssatz
- verschiedene Taktzahl pro Befehl
- Viele Befehlscode-Formate mit unterschiedlicher Länge
- Mikroprogrammwerk
- Vermischung von Verarbeitungs & Speicherbefehlen
- schwierig, unter CPI = 2 zu kommen

RISC reduced instruction set computer

- wenige, einfache Befehle
- orthogonaler Befehlssatz
- meist 1 Takt pro Befehl
- wenige Befehlscode-Formate mit einheitlicher Länge
- Direktverdrahtung
- Trennung von Verarbeitungs & Speicherbefehlen
- hohe Ausführungsgeschwindigkeit ($CPI \leq 1$)

MIPS

- Microprocessor without interlocked pipeline stages
- 32-bit Architektur/64-bit Erweiterung



Aufgaben der einzelnen Phasen

Befehlsholphase Lesen des aktuellen Befehls; separater zur Vermeidung von Konflikten mit Datenzugriffen

Dekodier & Register-Lese-Phase Lesen der Register möglich wegen fester Plätze für Nr. im Befehlswort

Ausführungs & Adressberechnungsphase Berechnung arithmetischer Funktion o. Adresse für Speicherzugriff

Speicherzugriffsphase Wird nur bei Lade & Speicherbefehlen benötigt

Abspeicherungsphase Speichern in Register, bei Speicherbefehlen nicht benötigt

Hazards

- resource hazards: Ressourcenabhängigkeiten
- data hazards: Datenabhängigkeiten

  Antidatenabhängig WAR (write after read)
  Ausgabeabhängig WAW (write after write)
  Datenabhängigkeit RAW (read after write)

- control hazards: Kontrollabhängigkeiten
  - Gleichheit der Register in ID-Stufe prüfen
  - Sprungziel in separatem Adressaddierer in ID berechnet

Sprungvorhersage

Einfache lokale Prädiktoren

- Vorhersage: bedingter Sprung genommen oder nicht
- Prädiktion anhand der Historie des betrachteten Sprungs
- Historie eines Sprungs wird mit 1 bis n Bits gepuffert

Einfache Sprungvorhersage (1 Bit)

- Branch prediction buffer oder branch history table
- Kleiner Speicher, der mit (Teil der) Adresse des Sprungbefehls indiziert wird
- Verwendet nur wenige untere Bits der Adresse
- Enthält 1 Bit: Sprung beim letzten Mal ausgeführt (taken) oder nicht (not taken)
- Prädiktion: Sprung verhält sich wie beim letzten Mal
- Nachfolgebefehle ab vorhergesagter Adresse holen
- Falls Prädiktion fehlerhaft: Prädiktionsbit invertieren
- Sprünge, gleicher Adressen im Indexteil, werden selber Zelle im branch prediction buffer zugeordnet
- Einfachste Art von Puffer
- Hat eine bestimmte Kapazität
- Kann nicht für alle Sprünge Einträge enthalten
- Reduziert branch penalty, wenn branch delay > Berechnung der Zieladresse mit branch p. buffer
- Prädiktion kann fehlerhaft sein
- Prädiktion kann von anderem Sprungbefehl stammen

Einführung von Tag Bits

- Sprünge, gleicher Adressen im Indexteil, in selber Zelle
- Tag: gültiger Eintrag, falls Tag-Bits gleich sind
- Fehlerrate 1-Bit-P in Schleifenkonstrukten doppelt so hoch wie Anzahl ausgeführter Sprünge

2 Bit Vorhersagen

- Änderung der Vorhersage nur, wenn 2 falsche Vorhersagen in Folge
- 2-Bit Branch-Prediction Buffer: Speicherung der Historie, Befehlsadressen als Zugriffsschlüssel

n-Bit Prädikator

- Verwendet n-Bit Zähler
- Sättigungsarithmetik (kein wrap around bei Überlauf)
- Kann Werte zwischen 0 und 2^{n-1} annehmen
- Wenn Zähler größer als Hälfte des Maximums (2^{n-1}): Vorhersagen, dass Sprung ausgeführt wird
- Zähler bei ausgeführtem Sprung inkrementiert und bei nicht ausgeführtem dekrementiert
- Praxis: 2-Bit Prädiktor ähnlich gut wie n-Bit Prädiktor

Korrelierende Prädikatoren

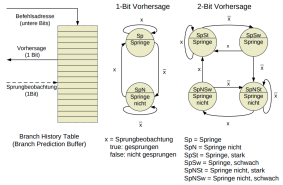
- Betrachtet nur Verhalten eines Sprungs; rein lokal
- Verbesserung durch Betrachtung anderer Sprünge
- erhält so korrelierenden/zweistufigen Prädiktor
- Prinzip: Aufgrund globaler Information wird einer von mehreren lokalen Prädiktoren ausgewählt
- Beziehen zur Vorhersage des Verhaltens Kontext-Information mit ein
- Prädiktor benutzt globale Kontext-Bits, um einen von mehreren lokalen Prädiktoren auszuwählen
- Betrachten wiederholte Ausführung des Codefragments

Zweistufiger Prädiktor

- Es existieren 2 lokale Prädiktoren, beide je 1-Bit
- Kontext: Letzter Sprung wurde (nicht) ausgeführt
- Anhand des Kontexts wird lokaler Prädiktor für die Vorhersage des aktuell betrachteten Sprungs ausgewählt
- Letzter Sprung ist i.a. nicht gleich aktuellem Sprung
- Notation des Prädiktorstatus:$< X > / < Y >$ mit
- $< X >$: Vorhersage, falls letzter Sprung not taken
- $< Y >$: Vorhersage, falls letzter Sprung taken
- $< X > \vee < Y >$ Vorhersagen: entweder T oder NT

(m,n)-Prädiktor

- Betrachtet das Verhalten der letzten m Sprünge, um aus 2^m vielen lokalen Prädiktoren einen n-Bit Prädiktor auszuwählen
- Höhere Vorhersagegenauigkeit
- Erfordert kaum Hardwareaufwand
- Sprunggeschichte in m-Bit Schieberegister gespeichert
- Vorhersagepuffer adressiert via Konkatenation von unteren Adressbits der Sprungbefehlsadresse



High Performance Befehlsdekodierung
reine Vorhersage eines Sprungs i.d.R. nicht ausreichend

- Befehlsstrom mit großer Bandbreite erforderlich
- Kontrollflussabhängigkeiten dürfen nicht wahrnehmbar sein
- Pufferung von Sprungzielen und nicht nur Vorhersage des Sprungverhaltens (branch target buffer)
- Integrierte Einheit für das Holen der Befehle
- Vorhersage von Rücksprungadressen (bei Prozeduraufruf)

Branch Target Buffer
5-stufige Pipeline, Auswertung von Sprungbedingungen in EX

- Branch delay von 2 Takten
- mit Sprungvorhersage (branch prediction buffer)
- Zugriff erfolgt in ID (Adresse des Sprungbefehls in IF bekannt)
- Nächste vorhergesagte Instruktion kann erst nach ID geholt werden
- Branch delay = 1, falls Prädiktion korrekt
- Mit Pufferung des Sprungziels
- Zugriff auf branch target buffer erfolgt in IF
- Verhalten wie „echter“ Cache, adressiert mit Sprungbefehlsadresse
- Liefert vorhergesagte Adresse als Ergebnis
- Keine Verzögerung, falls Prädiktion korrekt
- Zusätzliche Speicherung auch des Sprungziels
- bei geschickter Organisation bleibt das Fließband immer gefüllt
- Sprünge kosten dann effektiv keine Zeit; $CPI < 1$ mögl.

Eigenschaften

- Verzögerung durch Sprung vollständig vermieden, da bereits in IF Entscheidung über nächsten Befehlszähler getroffen
- Entscheidung allein auf Basis des PC getroffen wird, muss überprüft werden ob Adresse im Puffer
- Speicherung nur für Sprünge notwendig, die als ausgeführt vorhergesagt werden
- entsteht ursprüngliche Sprung-Verzögerung plus Aufwand zur Aktualisierung des Vorhersagepuffers

Integrierte Befehls-Hol-Einheit (IF Unit)
Insbesondere mit Blick auf multiple-issue Prozessoren eigene (autonome) funktionale Einheit für Befehlsholphase

- führt Befehlscodes in Pipeline ein
- Sprungvorhersage wird Teil der Befehlsholphase
- Instruction Pre-fetch: Insbes. um mehrere Befehle pro Takt liefern zu können, läuft Befehlsholen weiterer Dekodierung voraus
- Zugriff auf Befehlsspeicher: Bei mehreren Befehlen pro Takt mehrere Zugriffe erforderlich (bei Cache auf ggfs. mehrere cache lines). Werden hier koordiniert/geplant
- Befehlspuffer: Befehle können hier (lokal im Prozessor!) von Issue-Stufe nach Bedarf abgerufen werden

Vorhersage von Rücksprungadressen
Vorhersage indirekter Sprünge (d.h. bzgl. Basisadresse in Register)

- Hauptverwendung: Rückkehr aus Prozeduraufrufen
- MIPS: Prozeduraufruf per jal, Rückkehr per jr
- Vorhersage mit branch target buffer schlecht, da Aufruf aus unterschiedlichen Codeteilen heraus möglich
- Methode: (Stack-) Speicher für Rücksprungadressen
- Push bei Prozeduraufruf (call)
- Pop bei Rücksprung (return)
- Vorhersagequalität „perfekt“, wenn Stack-Puffer größer als maximale Aufruftiefe

Multiple-Issue-Architekturen

Mehrere Ausführungseinheiten

- Weitere Leistungssteigerung $CPI < 1$
- Mehrere Befehle pro Takt ausgeben
- Zwei Grundtypen von multiple-issue Prozessoren
  - Superskalar: var. Anzahl von Befehlen pro Takt
  - VLIW/EPIC: Feste Anzahl von Befehlen ausgegeben, definiert durch Befehlscode (Planung der Issue-Phase durch Compiler)

Superskalar

- IF holt 1-n Befehle von Instruction Fetch Unit
- Befehlsgruppe, die potentiell ausgegeben werden kann = issue packet
- Konflikte bzgl. Befehlen im issue packet werden in Issue-Stufe in Programmreihenfolge geprüft
- Befehl ggfs. nicht ausgegeben (und alle weiteren)
- Aufwand für Prüfung in Issue-Stufe groß!
- Wegen Ausgewogenheit der Pipeline-Stufen ggfs. Issue in mehrere Stufen unterteilen = nicht-trivial
- Parallele Ausgabe von Befehlen limitierender Faktor superskalarer Prozessoren

MIPS mit statischem Scheduling

- Annahme: 2 Befehle pro Takt können ausgegeben werden (1x ALU, Load/Store plus 1x FP)
- Einfacher als 2 beliebige Befehle (wegen „Entflechtung“)
- 2 Befehlsworte holen (64-Bit Zugriff, komplexer als 1 B.)
- Prüfen, ob 0/1/2 Befehle ausgegeben werden können
- Befehle ausgeben an korrespondierende funk. Einheiten
- Prüfen auf Konflikte durch Entflechtung vereinfacht
- Integer und FP-Operationen nahezu unabhängig
- Abhängigkeiten nur bei Speichertransfers möglich (von Integer-ALU für FP ausgeführt)
- Leistungssteigerung nur bei geeignetem Anteil von FP-Operationen und Verflechtung durch Compiler

Dynamisches Scheduling
in-order-execution

- Jeder Befehl, der aus der Instruction fetch-Einheit kommt, durchläuft Scoreboard
- Wenn für Befehl alle Daten/Operanden bekannt sind und Ausführungseinheit frei ist, wird Befehl gestartet
- Alle Ausführungseinheiten melden abgeschlossene Berechnungen dem Scoreboard
- Scoreboard erteilt Befehlen die Berechtigung zum Abspeichern von Ergebnissen und prüft, ob dadurch neue Befehle ausführbereit werd
- Zentrale Datenstruktur: Scoreboard (für Befehlsstatus)
- load/store-Architektur
- mehrere funktionale Einheiten
- Scoreboarding für MIPS nur sinnvoll wenn:
- für FP-Pipeline (Operationen mit mehreren Taktzyklen)
- mehrere funktionale Einheiten (zB: 2xMult, Div, Add, Int)

Verfahren von Tomasulo

- erlaubt bei Ausgabe-/Antidatenabhängigkeiten Reihenfolge zu vertauschen
- Umbenennung der Register
- verschiedenen Benutzungen eines Registers werden verschiedene Speicherzellen zugeordnet
- Jeder funktionalen Einheit wird eine Reservation Station zugeordnet
- Reservation Stations enthalten die auszuführende Operation und die Operanden/tags des Operanden
- Sind alle Operanden bekannt und ist die funktionale Einheit frei, so kann die Bearbeitung beginnen
- Am Ende der Bearbeitung wird das Ergebnis von allen Einheiten übernommen, die das Ergebnis benötigen
- Verteilen der Daten erfolgt vor der Abspeicherung im Registerspeicher
- Aus den tag bits geht hervor, aus welcher Einheit der Operand kommen muss
- Registeradressen werden dynamisch auf größere Anzahl von Plätzen in den Reservation Stations abgebildet
- Performance-Beschränkungen wegen weniger Register werden so umgangen

Register Renaming

- Verwendung temporärer Register für (logisch) neue möglicherweise interferierende Belegung
- Alle Namenskonflikte durch Umbenennung auflösbar
- Wichtige Hardwarestruktur: Reservation Stations
- Zugeordnet zu funktionalen Einheiten (pro Einheit)
- Puffern Operanden für Befehle (sobald verfügbar)
- Müssen nicht aus Registern gelesen werden
- Ausstehende Operanden verweisen auf Reservation Station, die Eingabe bereitstellen
- Bei aufeinander folgenden Schreibzugriffen auf Register: Nur letzter für Aktualisierung des Inhalts verwendet
- Wichtige Eigenschaften der Verwendung von Reservation Stations anstelle des zentralen Registersatzes
- Konfliktdetektion und Ausführungskontrolle verteilt
- Informationen in Reservation Stations bei den funktionalen Einheiten bestimmen, wann Ausführung eines Befehls möglich ist
- Ergebnisse werden direkt zu den funktionalen Einheiten (in jeweiliger Reservation Station) weitergereicht
- Erweiterte Form des Forwarding
- Realisiert implizit Register Renaming durch gemeinsamen Ergebnisbus (common data bus)

Multiple-Issue mit dynamischem Scheduling

- Nachteil von statischem Scheduling: Latenzzeiten werden ca. mit Länge des issue packets skaliert
- Längere Verzögerung für Load/Stores bzw. Branches
- Lösung: Erweiterung des Tomasulo-Algorithmus auf Multiple-Issue
- Sequentielles Ausgeben mehrerer Befehle an Reservation Stations innerhalb eines Taktes
- oder „Verbreiterung“ der Ausgabe-Logik (issue logic) zur Behandlung mehrerer Operationen parallel

VLIW
Very Long Instruction Word

- Befehlszuordnung und Konfliktvermeidung durch Compiler
- Compiler muss Zeitbedarf der Speicherzugriffe in Befehlsplanung einbeziehen
- Befehlsstrom mit Tupel von Befehlen
- flexibel bei Reaktion auf Laufzeitereignisse
- VLIW hat festes Befehlsformat; höhere Codedichte

- Forwarding Hardware - nach EX können Daten vom nächsten Befehl genutzt werden
- WB erfolgt erst darauf
- Datenabhängigkeitsproblem wird verringert
- MUX nutzt eher Pufferdaten als Registerdaten
- verschiedene parallele Ausführungseinheiten
- Verteilung von Maschinencode direkt vom Befehlswort im Speicher vorgegeben
- für jede Ausführungseinheit dezidierte Anweisungen
- meist für stark parallelisierbare Aufgaben verwendet
- Vorteile:
  - parallele Architektur des Prozessors kann während der Programmerstellung zur Optimierung genutzt werden
  - Keine aufwendige Prozessorhardware zur Befehlsverteilung erforderlich
  - Ausführungszeiten sind im wesentlichen bekannt
- Nachteile:
  - Aufwendigere Compiler
  - Schlechte Prozessorauslastung bei ungünstigem Code
  - Rekompilierung für den Prozessor erforderlich
  - Größerer Speicherbedarf, wenn Code nicht parallelisiert werden kann

EPIC Explicitely Parallel Instruction Computing = IA64

- im wesentlichen Prinzip des VLIW-Prozessors
- Umsortieren der Befehle und Auflösung der Abhängigkeiten wird durch Compiler durchgeführt
- Hauptnachteil: Neukompilierung erforderlich
- Keine statische Aufteilung auf Funktionseinheiten
- Effizienteres Befehlswort
- Keine Verwendung von zwangsweise NOPs

verschiedene Ansätze, um die Prozessorlogik zu vereinfachen

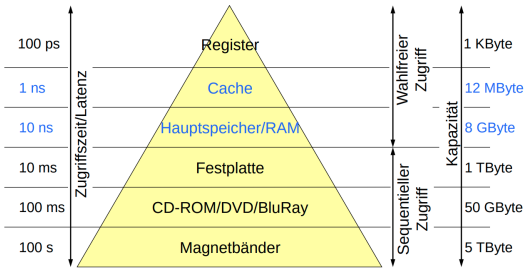
1. Bedingte Befehlsverarbeitung
   - Befehl abhängig von Statusbit ausgeführt
   - Sprungvorhersage kann bei einfachen if-then-else Zweigen entfallen
   - then/else Befehle parallel, nur einer ausgeführt
2. Statische Sprungvorhersage (Compiler)
3. Die Optimierung wird dem Compiler überlassen
4. Spekulatives Laden von Operanden
   - Möglichst geringe Wartezeit auf Operanden
   - Schon im Compiler werden entsprechende Ladebefehle vorgezogen

Simultaneous Multithreading (SMT)

- Modellprozessor I (2-fach Superskalar)
- Modellprozessor II (2-fach Out-of-Order)

Speicherhierarchie

- Große Speicher sind langsam
- Anwendung verhalten sich üblicherweise lokal
- Häufig benötigte Speicherinhalte in kleinen Speichern



Speicherarchitekturen

Adresspipelining

- Aufteilen des Speicherzugriffs in mehrere Phasen
- parallele gestaffelte Abarbeitung dieser Phasen für mehrere Speicherzugriffe
- Adresse auslesen/dekodieren; Daten mit Prozessor

Lesezugriff auf Speicher

- Matrixaufbau eines Speichers
- Aufteilen der Speicheradresse in Zeilen- und Spalten
- Dekodierung der Zeilenadresse bestimmt Select-Leitung
- Komplette Zeile wird in den Zeilenpuffer geschrieben
- Dekodierung der Spaltenadresse bestimmt Datenwort

Speicher Interlacing

- Speicheraufteilung in mehrere physische Bände
- Adressen nicht kontinuierlich in den Bänden, sondern wechseln von Band zu Band
- nahezu gleichzeitiges Benutzen der Daten möglich (Daten-/Adressbus verhindern Parallelität)

Burst Mode Blocktransfer

- Auslesen des kompletten Zeilenpuffers durch automatisches Inkrementieren der Spaltenadresse
- Prozessor gibt eine Adresse, Speicher liefert n Datenworte (Adr, Adr+1,..., Adr+n-1)
- falls folgende Datenworte genutzt werden, war für n Datenworte nur 1 Speicherzugriff (Zeit) nötig

Typischer DRAM-Speicher

- Adressleitungen werden i.d.R. gemultiplext
- gleiche Adressleitungen werden einmal zur Auswahl der Zeile verwendet, dann zur Auswahl der Spalte
- Einsparung von Leitungen, gerade für große Speicher
- Steuerleitungen RAS/CAS codieren
- RAS (Row Address Strobe): Bei fallenden Flanke auf RAS ist anliegende Adresse Zeilenadresse
- CAS (Column Address Strobe): Bei fallenden Flanke auf CAS ist anliegende Adresse Spaltenadresse
- Zeilenadressdecoder liefert Select-Leitung für eine Zeile
- Komplette Zeile wird in einen Zwischenpuffer übernommen und zurückgeschrieben
- Nur 1 Transistor und 1 Kondensator pro Speicherzelle, statt 6 Transistoren bei SRAM
- Integrationsdichte Faktor 4 höher als bei SRAMs
- Weniger Platzbedarf aber Langsamerer Zugriff wegen Zwischenspeicherung und Auffrischung
- während Auffrischung kann nicht zugegriffen werden
- Hoher Energieverbrauch bei Aktivität/Inaktivität
- Ladungsverlust Ausgleich durch periodische Auffrischung

Cache Speicher

- kleiner, schneller prozessornaher Speicher
- Puffer zwischen Hauptspeicher und Prozessor
- CPU weiß nicht dass Cache zwischengeschaltet ist
- es wird immer zuerst im Cache nachgeschaut (kostet Zeit)
- 90% der Zeit verbringt ein Programm in 10% des Codes
- Cache besteht aus Cache Tabelle

voll assoziativ Adressvergl. der kompletten Adresse
direct-mapped Adressvergleich nur über Teiladresse
mehr-wege-assoziativ mehrere Adressverg. parallel

- Schreibstategien

Write Back Cache sammelt Schreibvorgänge und aktualisiert Speicher nach Anweisung
Copy Back Rückschreiben erfolgt erst, wenn Cache-Zeile bei Miss verdrängt wird
Write Through Daten werden sowohl im Cache als auch im Hauptspeicher aktualisiert

- Speicherverwaltung mit memory management günstiger vor dem Cache
- cache hit: benötigte Daten im Cache vorhanden
- cache miss: Zugriff auf den (Haupt-) Speicher
- Such-Einheit im Cache: Cache-Zeile (cache line)
- Blockgröße ist Anzahl der Worte, die im Fall eines cache misses aus Speicher nachgeladen werden
- Verbindung Speicher ↔ Cache ist so entworfen, dass Speicher durch zusätzliches Lesen nicht langsamer wird
- Methoden dazu:
  - Schnelles Lesen aufeinanderfolgender Speicherzellen (Burst Mode)
  - Interleaving (mehrere Speicher ICs mit überlappenden Zugriffen)
  - Fließbandzugriff auf den Speicher (SDRAM)
  - Breite Speicher übertragen mehrere Worte parallel
- Ersetzungs-Strategien

Zufall zu ersetzende Block zufällig ausgewählt
FIFO älteste Block ersetzt
LRU (least recently used) längsten nicht zugegriffen
LFU (least frequently used) am seltensten gelesene
CLOCK Zeiger im Uhrzeigersinn zeigt Eintrag

- Trefferquote $T = \frac{N_{CacheZugriffe}}{N_{CacheHit}}$

Spezialrechner

Einchiprechner

- geringer Stromverbrauch, Wärme
- relativ geringer Befehlsdurchsatz
- einfacher Befehlssatz
- komplexer Rechner auf einem Chip (Ram/Rom intern)
- an Problem angepasst
- so Leistungsfähig wie nötig
- Anwendung: einfache Steuer-/Regelungsaufgaben

Digital-Signal-Prozessoren

- hohe Leistung, u.a. sich wiederholende, numerisch intensive Aufgaben
- pro Befehlszyklus kann man ausführen
  - mehrere ALU Funktionen (+Shifter)
  - eine/mehrere MAC-Operationen
  - ein/mehrere Speicherzugriffe
  - spezielle Unterstützung effiziente Schleifen
- Die Hardware enthält
  - Eine oder mehrere MAC-Einheiten
  - On-Chip- und Off-Chip-Speicher mit mehreren Ports
  - Mehrere On-Chip-Busse
  - Adressgenerierungseinheit
  - größerer ROM, RAM (mit Stack), Cache
  - externes Interface für Speicher und E/A
  - DMA-Coprozessor
  - mehrere universelle parallele E/A-Ports
  - AD-Wandler meist nicht on-chip
- DSP's benutzung häufig VLIW
- Anwendung: Signal-/Bildverarbeitung

Pipeline Prozessoren

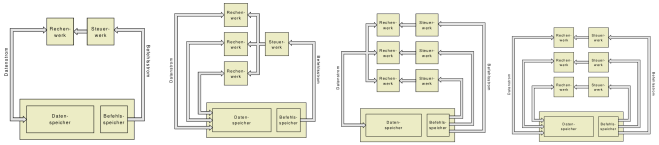
- Aufteilung eines Befehls in Teilbefehle
- parallele Abarbeitung verschiedener Teilbefehle
- Problem: bedingte Sprünge (unvorhergesehen)
  - LSG: Pipeline um 2 Schritte verzögern
  - LSG: Sprungzielspekulation
- Problem: Datenabhängigkeit
  - LSG: Pipeline um 2 Schritte verzögern
  - LSG: Out-of-Order Prinzip nutzen
  - LSG: Forwarding Hardware
- Superpipelining - noch mehr Befehlsaufteilung

Skalare Prozessoren

- Mikroarchitektur
- Befehlszuordnung und Konfliktvermeidung geschieht durch Hardware
- Speicherzugriffe automatisch von Load/Store Einheit
- Befehlsstrom mit einfachem Befehl an EX-Einheit
- bessere Reaktion auf Laufzeitereignisse
- Spekulation möglich
- Superskalar (≥ 2 EX Einheiten)
  - Befehle in Befehlsfenster gespeichert
  - Zuordnungseinheit wählt Befehl aus, die parallel verarbeitet werden können

Multiprozessorarchitekturen nach Flynn

	Ein Datenstrom	mehrere Datenströme
ein Befehlsstrom	SISD	SIMD
mehrere Bs	MISD	MIMD



Kopplung

enge Kopplung (shared memory)

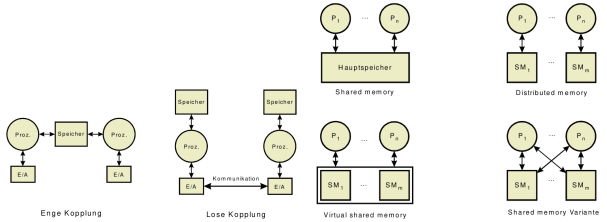
- parallelzugriff in Datenbreite des Prozessors
- schneller Datenaustausch
- neu lokal benachbarte Rechner
- aus Sicht des Prozessors gemeinsamer Speicher
- Arbites - Zuteiler für Busanforderungen

lose Kopplung (distributed memory)

- meist seriell 1 bit breit
- langsamer da über Kommunikations-schnittstelle
- auch global verteilte Rechner
- verschiedene Speicher dem Prozessor bewusst

Kopplung verändern

- Wartezeit auf Speicher
- Kommunikationsaufwand
- Kommunikationsfähigkeit



Verbindungsnetzwerke

Linie $KL_{max} = n - 1$
Ring $KL_{max} = \frac{n}{2}$
Torus 2D $KL_{max} = \sqrt{n} - 1$
Torus 3D $KL_{max} = \frac{3}{2}\sqrt[3]{n} - 1$
Hypercube $iD = i$
Gitter $iD = i * \sqrt[i]{n} - 1$

Out-of-Order Architektur

- statt Pipeline bei Datenabhängigen Befehlen um 2 Schritte verzögern, datenunabhängige Befehle einschieben
- möglichst ständige Auslastung aller EX Einheiten

Cache(daten)-Kohärenz

- Kohärenz: welcher Wert wird beim Lesen abgeliefert
- Bezug auf Lesen und Schreiben ein- und derselben Speicherzelle
- Definition: Ein Speichersystem heißt kohärent, wenn
  - geschriebene Werte werden wieder gelesen
  - Schreibvorgänge derselben Zelle serialisiert
- Lösung des I/O-Problems: Zuordnung einer I/O-Einheit zu jedem Prozessor
- Hardware-Lösung: Aufwändig, schlechte Lokalität der Daten
- Gemeinsamer Cache für alle Prozessoren: Hoher Hardware-Aufwand, geringe Effizienz
- Unterscheidung in cacheable/non-cacheable Daten: Hoher Aufwand

Snooping-Protokolle

- Caches aller Prozessoren beobachten alle Datenübertragungen von jedem Cache zum Hauptspeicher
- Voraussetzung: broadcastfähiges Verbindungsnetzwerk

Write Invalidate Verändern eines Blocks im Speicher führt zur Invalidierung aller Cache-Kopien mit der gleichen Adresse

Write Update/Broadcast Verändern eines Blocks im Speicher führt zur Modifikation aller anderen Cache-Blöcke mit der gleichen Adresse

Copy-Back

- Copy-Back Caches führen zur temp. Inkonsistenz
- Lösung: exklusives Eigentumskonzept durch Zustandsgraph pro Cache-Block
- MESI (Modified, Exclusive, Shared, Invalid)
- Mischung zwischen Write-Through und Copy-Back

MESI

Zustände	I Invalid	E Exclusive	M Modified	S Shared
Cacheinhalt gültig?	nein	ja	ja	ja
Speicherinhalt aktuell?	-	ja	nein	ja
Kopie in anderen Caches?	-	nein	nein	möglich
Buszyklus beim Cache-Schreiben	global	lokal	lokal	global

Bedingungen		
Read Hit Write Hit	Treffer beim lokalen	Lesen Schreiben
Snoop Hit (Read) Snoop Hit (Write)	Treffer beim fremden	Lesen Schreiben
Read Miss Write Miss	Fehltreffer beim lokalen	Lesen Schreiben
shared not shared	mit ohne	Snoop Hit in anderem Cache

Aktionen	
Line Fill	Speicherinhalt in Cache kopieren
Copy Back	Cacheinhalt in Speicher kopieren
Read for Ownership	Snoop Hit (Write) in anderen Caches erzwingen

Bus-Operationen

Bus Read Prozessor liest Wert eines Speicherblocks
Bus Read Exclusive Prozessor überschreibt Wert eines Speicherblocks
Flush Prozessor P_i hat Speicherblock alleinig in seinem Cache, ein anderer Prozessor P_j aber lesend oder schreibend auf diesen Block zugreift

Steuersignale

- Invalidate-Signal: Invalidieren des Blocks in den Caches anderer Prozessoren
- Shared-Signal: Signalisierung, ob ein zu ladendes Datum bereits als Kopie im Cache vorhanden ist
- Retry-Signal: Aufforderung von Prozessor P_i an Prozessor P_j, das Laden eines Datums vom Hauptspeicher abzubrechen, da der Hauptspeicher noch ein altes, ungültiges Datum besitzt und vorher aktualisiert werden muss. Das Laden durch P_j kann danach wiederholt werden.

Bewertung von Snooping-Protokollen

- Leichte Implementierbarkeit bei Bus-basierten Shared Memory Systemen
- Snooping skaliert bei Bussen jedoch nicht
- Bei vielen beteiligten Prozessoren sinkt die effektive Bandbreite des Busses, da überproportional viele Invalidierungsnachrichten per Broadcast über den Bus gehen
- Punkt-zu-Punkt Netzwerke sind skalierbar, jedoch ist die Implementierung von Broadcasts hier aufwändig
- Für Snooping-Protokolle daher oft ungeeignet

Directory-Protokolle

- Nur wenige Prozessoren teilen sich die gleichen Daten in vielen Anwendungen
- Directory-Protokolle nutzen Lokalitätsinformationen, um die Anzahl an Invalidierungsnachrichten zu minimieren
- Nachrichten gehen nur an Prozessoren, die eine Kopie des Cache-Blocks besitzen
- Directory-Protokolle skalieren daher auch für Netze ohne Broadcast-Fähigkeit
- Presence Flag Vector: Im Hauptspeicher abgelegter Bit-Vektor für jeden einzelnen Speicherblock (1 Bit pro Prozessor/Cache + Statusbits (dirty, modified))
- Problem: Wachstum des Speicherbedarfs linear mit Anzahl der Prozessoren