Karlsruher Institut für Technologie

Institut für Prozessdatenverarbeitung und Elektronik

Institut für Technik der Informationsverarbeitung

2. Zielarchitekturen

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

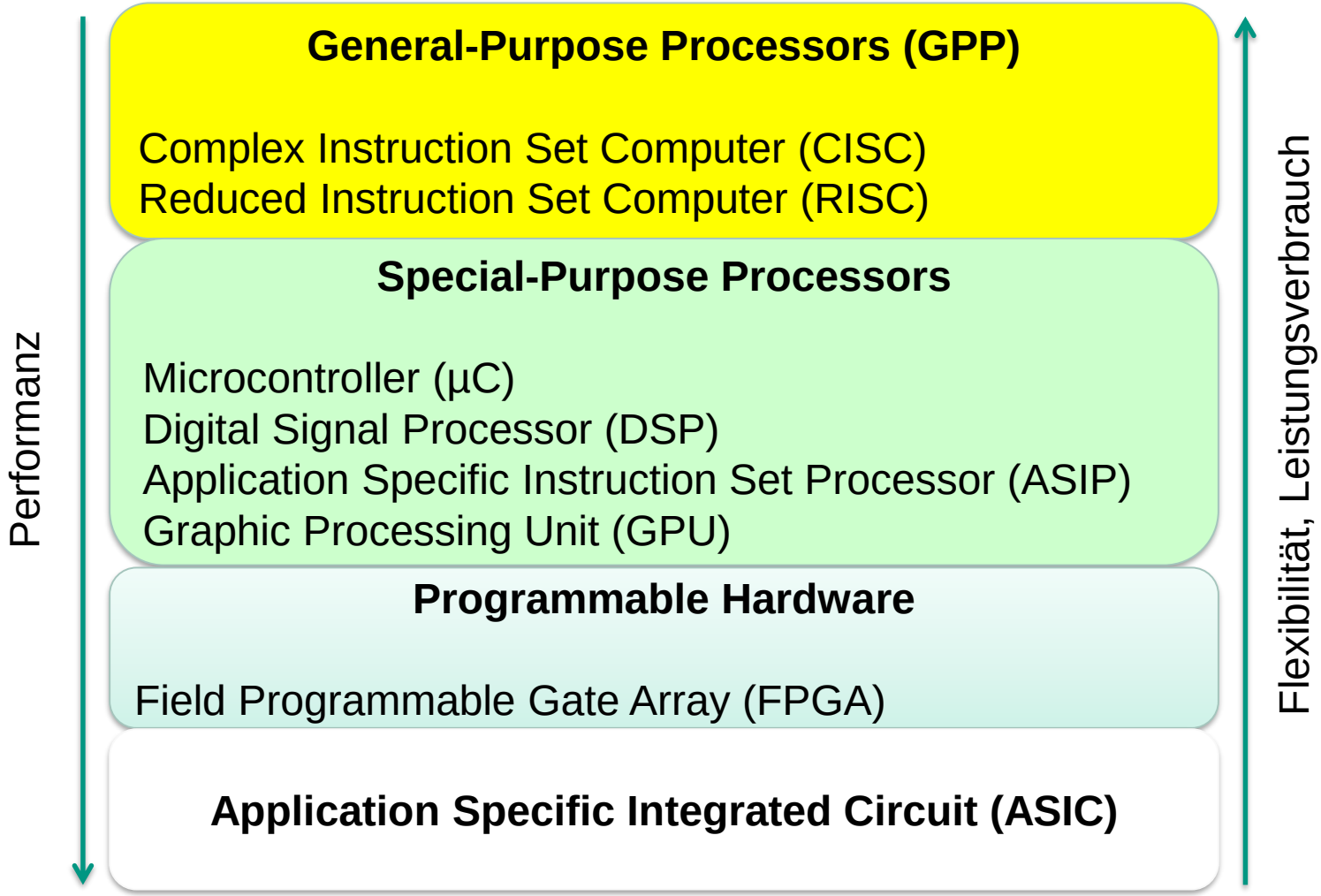
Hardware/Software Co-Design

Inhalt

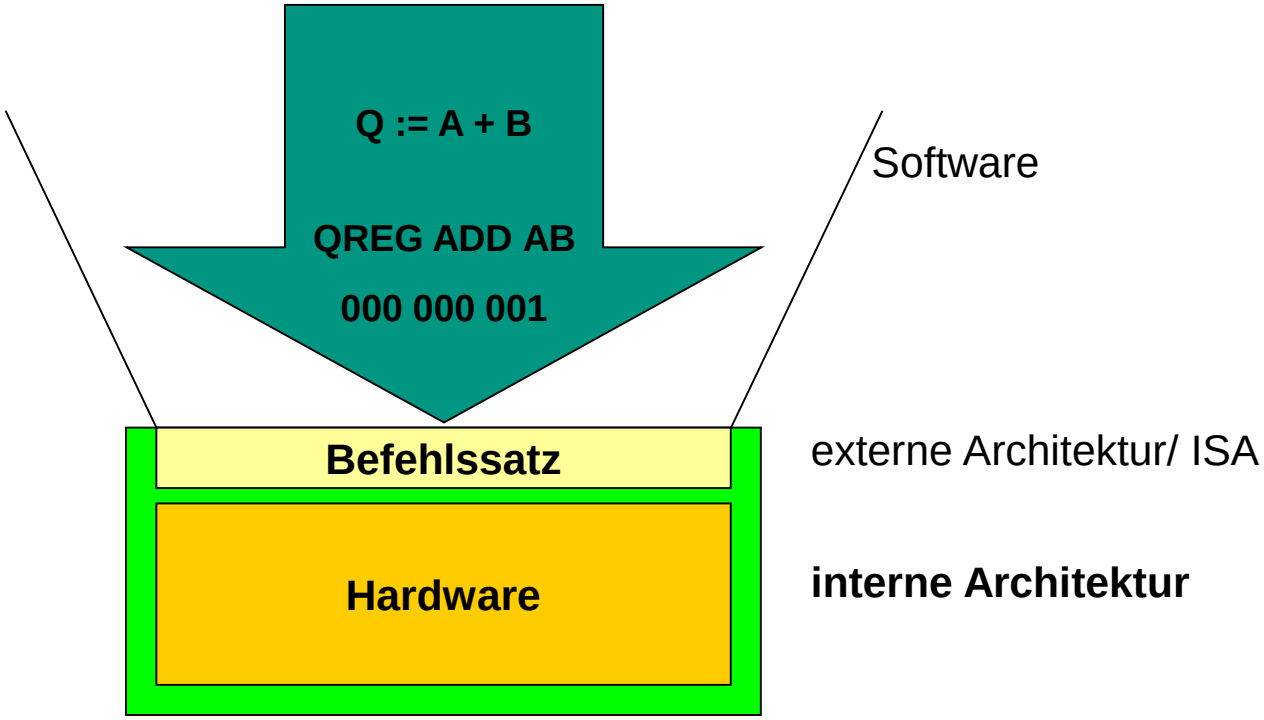
Inhalt

- 2.4 Spezialprozessoren
  - 2.4.1 Mikrocontroller (µC)
  - 2.4.2 Digitale Signalprozessoren (DSPs)
  - 2.4.3 Grafik Prozessoren (GPU)
- 2.5 Bus, Network-on-Chip (NoC) & Multicore
- 2.6 Anwendungs-spezifische Instruktionssatz Prozessoren (ASIP)
- 2.7 Field Programmable Gate Arrays (FPGA)
- 2.8 System-on-Chip (SoC)

2 Vergleich der Zielarchitekturen



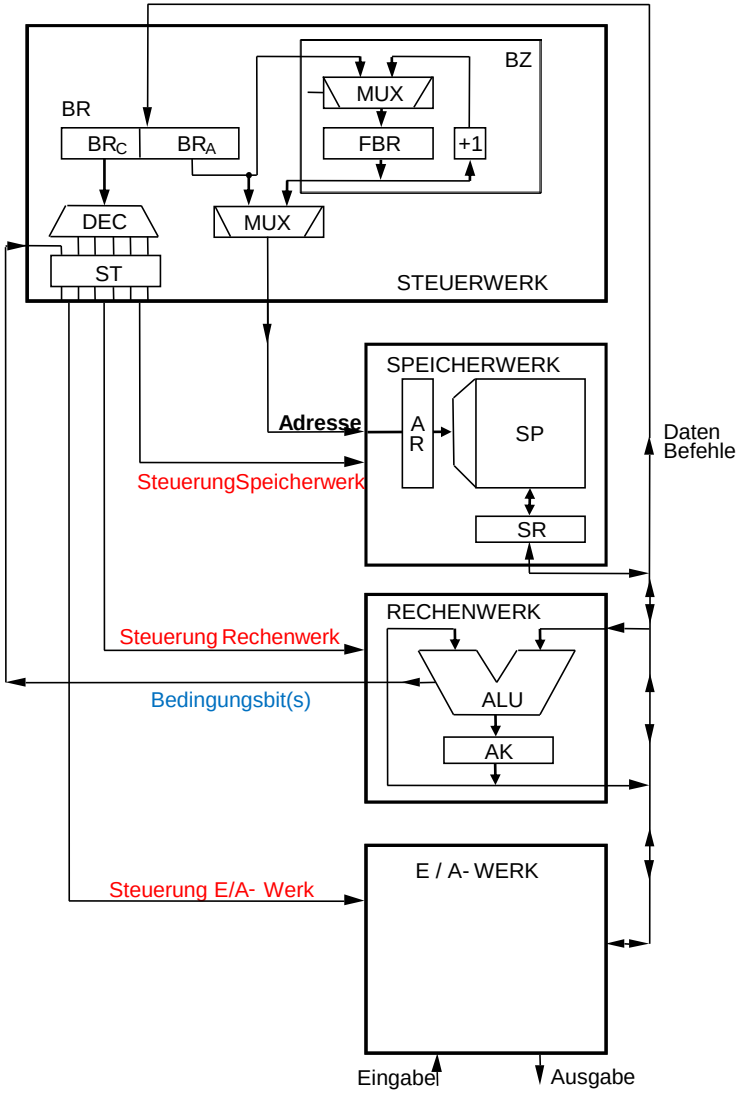
2 Rechnerarchitekturen



- Interne Architektur (Prozessor Mikroarchitektur) definiert die interne Struktur des Prozessors
  - Verschiedene interne Architekturen können dieselbe externe Architektur aufweisen
  - Computerfamilie

2.1 Aufbau eines einfachen Prozessors

- Steuerwerk
  - BR: Befehlsregister
    - BR_C: Codeteil
    - BR_A: Adressteil
  - ST: Register für Steuerbits
  - BZ: Befehlszähler
- Speicherwerk
  - AR: Adresszähler
  - SP: Speicher
  - SR: Speicherregister
- Rechenwerk
  - ALU: Arithmetische-, logische Einheit
  - AK: Akkumulator
- E/A-Werk
  - E/A: Eingabe/Ausgabe



2.1.1 Steuerwerk

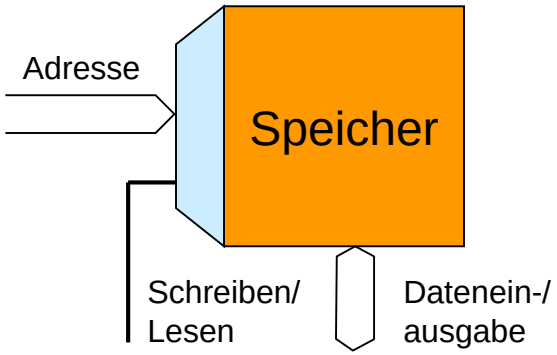
- Koordination der Komponenten des Prozessors:
  - Versorgung mit Daten und Adressen
  - Selektion von Geräten
  - Auswahl von Operanden und Operationen

- Ursprünglich Steuerung durch den Bediener, später Automatisierung durch zeitliche Abfolge von Binärinformationen

- Formalisierung der Anweisung nötig:
  - Befehl: (Maschinenlesbare) Anweisung, aus der hervorgeht, welche Operation mit welchen Operanden durchgeführt werden soll
  - Befehlsformat: Festlegung, wie die für einen Befehl notwendigen Angaben dargestellt werden

2.1.2 Speicherwerk

- Im einfachsten Fall nur eine Reihe von Registern zur Aufbewahrung von Operanden, Bedingungsvektor und Ergebnis.
- I.A. jedoch ein mehr oder weniger umfangreicher, adressierbarer Schreib/Lese-Speicher.
- Ermöglicht es, während der Verarbeitung eine große Zahl von Operanden und (Zwischen-)Ergebnissen bereitzuhalten, um so nicht für jede Operation das Ein-/ Ausgabewerk benutzen zu müssen.
- Die Zahl der verfügbaren Speicherzellen richtet sich nach der Breite des Adressvektors.



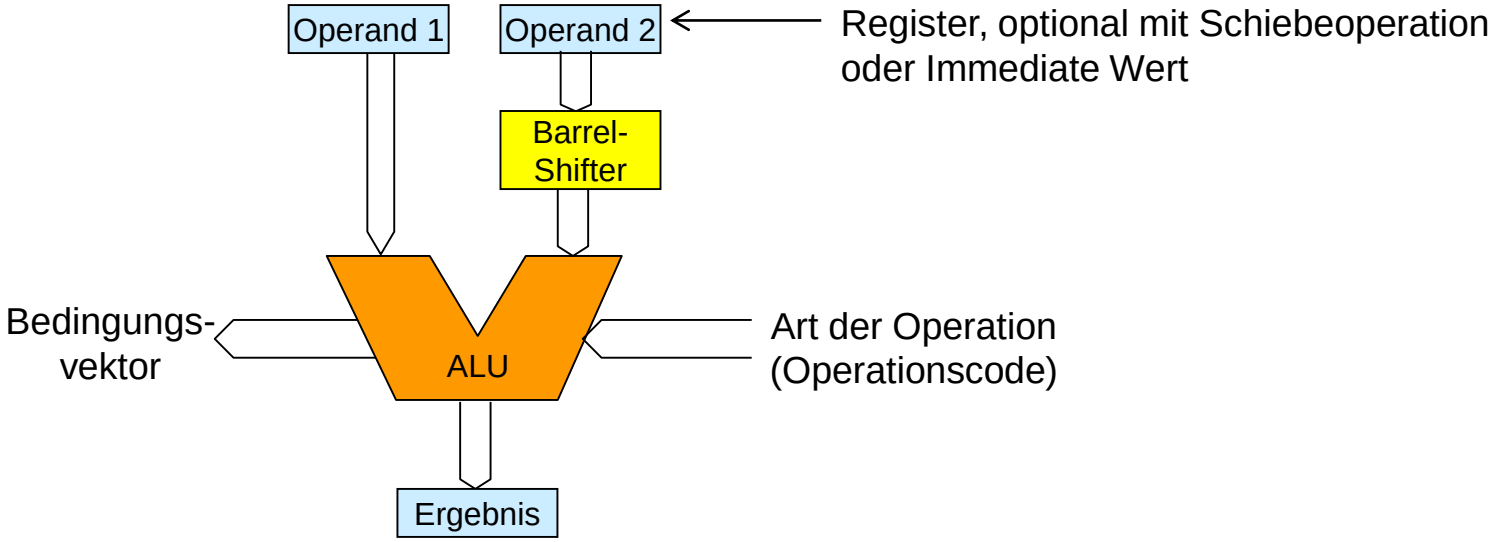
2.1.3 Rechenwerk

- Verarbeitung der Daten
- Stellt einen Satz von Basisfunktionen aus dem Bereich der numerischen und logischen Verknüpfungen zur Verfügung
- Beispiele für ein einfaches Rechenwerk:
  - Arithmetisch
    - Addition, Subtraktion
  - Logisch
    - Konjunktion, Disjunktion, Vergleich
    - Negation
  - Komplementbildung
  - Links-/Rechtsverschiebung
  - Links-/Rechtsrotation

2.1.3 Rechenwerk – Arithmetisch-logische Einheit

- Üblicherweise Beschränkung auf Funktionen mit zwei Operanden bei gegebener maximalen Stellenzahl
  - Genauigkeit (Bitbreite der Architektur)

- Einzelne Merkmale eines Operanden oder Ergebnisses (z.B. Wert gleich Null) können als Binärwerte in einem sogenannten Bedingungsvektor („flags“) zur Verfügung gestellt werden
  - Carry, Negative, Zero, Overflow

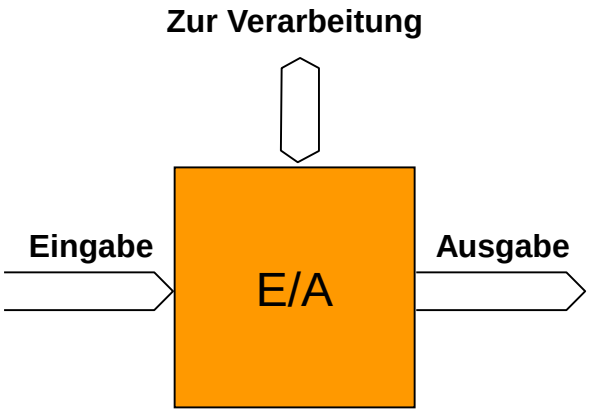


2.1.4 Eingabe-/Ausgabewerk

- Kommunikation mit dem Benutzer

- Eingabe:
  - Datenübernahme von extern angeschlossenen Geräten
  - Umsetzung in eine geeignete, normierte Darstellung zur weiteren Verarbeitung

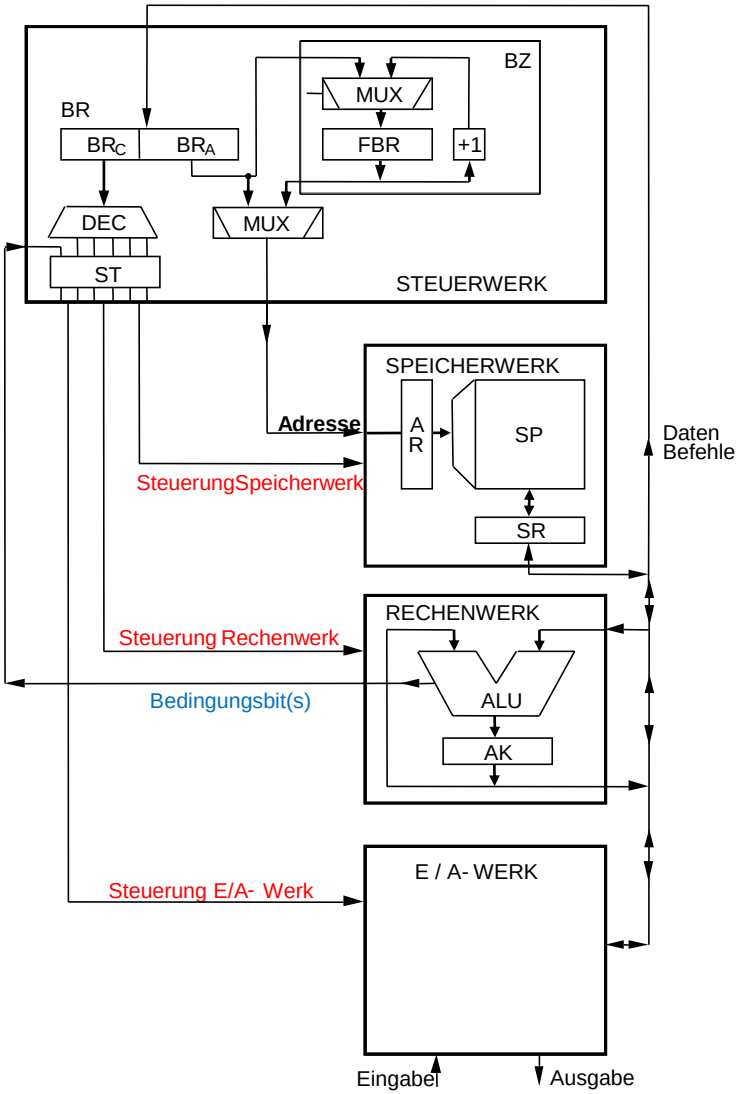
- Ausgabe:
  - Aufbereitung des Datenformats
  - Weiterleitung an externe Ausgabegeräte
  - Eventuell Auswahl des Ausgabegerätes



2.1.5 Von-Neumann-Zyklus (Arbeitsweise)

1. FETCH: Befehl aus dem Speicher in den Prozessor holen (lesender Zugriff)
2. DECODE: Befehl im Steuerwerk dekodieren
3. FETCH OPERANDS: gegebenenfalls Operanden aus dem Speicher oder der Peripherie in den Prozessor holen (lesender Zugriff)
4. EXECUTE: Befehl im Rechenwerk ausführen, ggf. Veränderung des Sprungzählers
5. WRITE BACK: gegebenenfalls Ergebnis in den Speicher oder die Peripherie schreiben (schreibender Zugriff)

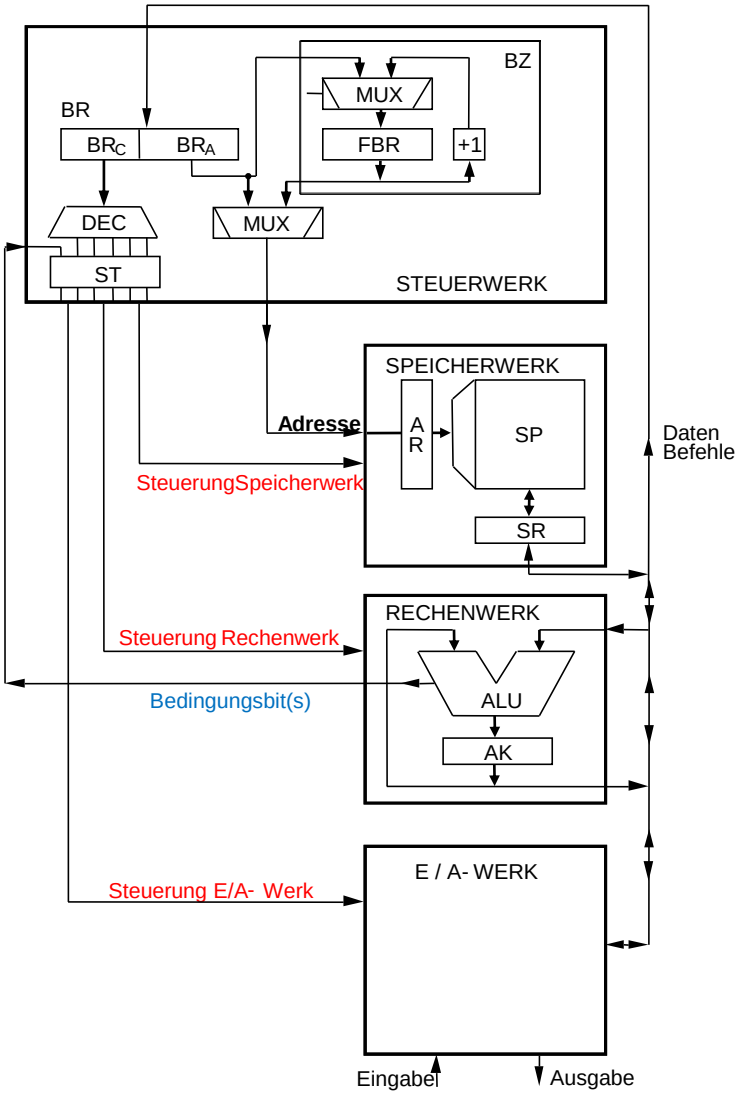
Weiter bei Schritt 1



2.1.6 Befehlstypen

Befehle:

1. Transportbefehle
   - Externer Transport (Arbeitsspeicher, E/A-System)
   - Interner Transport (Register-Register)
2. Verarbeitungsbefehle
   - Arithmetische und logische Befehle
   - Schiebebefehle
   - Vergleichsbefehle
3. Sprungbefehle
4. Systembefehle (z.B. Unterbrechungsbearbeitung, Speicherverwaltung)



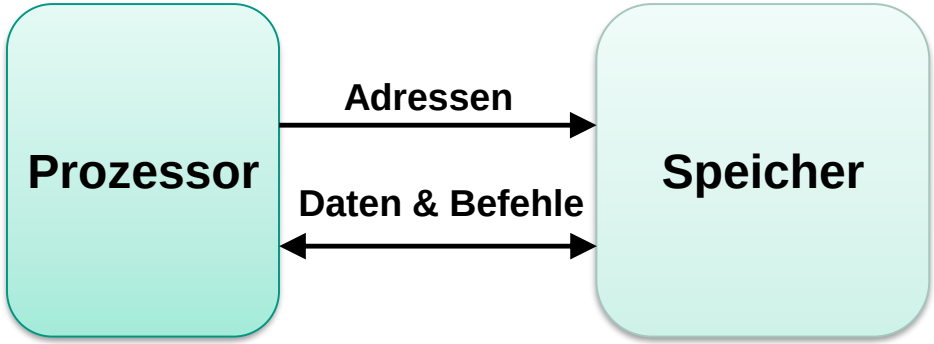
Abkürzungen:

SP	Speicher
AR	Adressregister
SR	Speicherregister
ALU	Arithmetik / Logik-Einheit
AK	Akkumulator
E/A	Ein-/Ausgabe-Werk

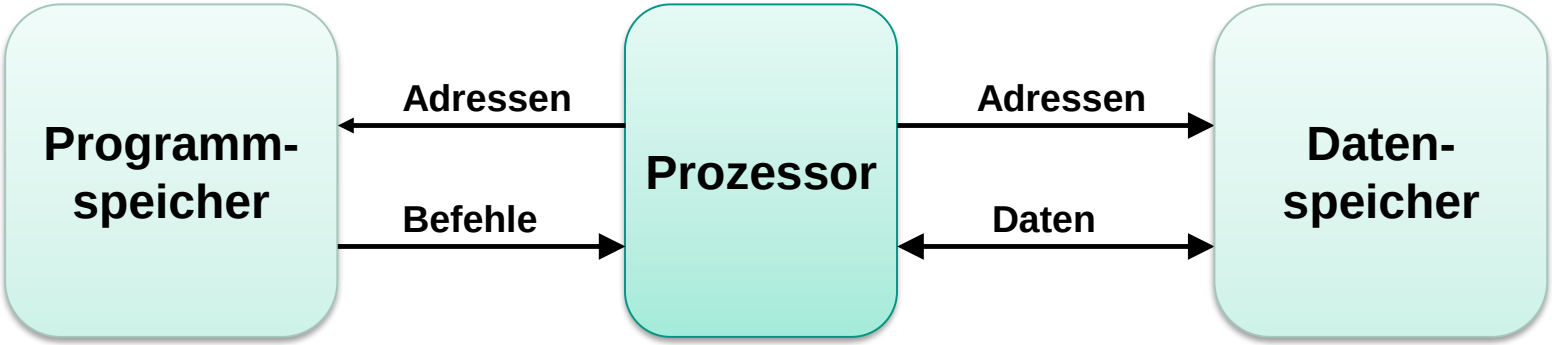
BR	Befehlsregister
	BR_C: Codeteil
	BR_A: Adressteil
ST	Register für Steuerbits
BZ	Befehlszähler

2.1.7 Speicheranbindung

- Von Neumann-Architektur:



- Harvard-Architektur:

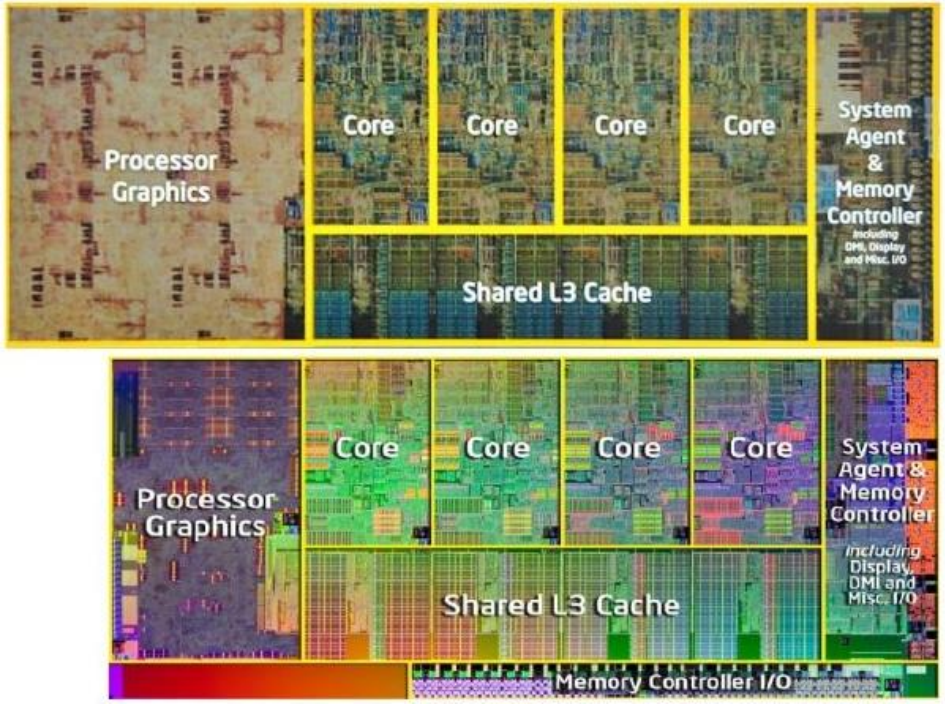


2.1.6 Beispiel: Intel Sandy Bridge vs. Ivy Bridge

- 1,16 Mrd. vs 1.4 Mrd. Transistoren
- 32nm vs. 22nm Prozess
- 212 mm^2 vs 160 mm^2 Die Größe

- QuadCore CPU
  - 3,5 GHz (3,9 GHz Turbo Mode)

- 4x32 kB I-&D-Cache (8-Wege)
- 4x256 kB L2 Cache (8-Wege)
- 8 MB L3 Cache (16-Wege)



http://www.itproportal.com/2012/04/24/intel-ivy-bridge-architecture-breakdown/

2.1.6 Beispiel: Intel Ivy Bridge

		intel inside CORE i7	intel inside CORE i7	intel inside CORE i7
Brand				
Processor Number		i7-3720QM	i7-3820QM	i7-3920XM
Price		$378	$568	$1096
TDP		45W	45W	55W
Cores/ Threads		4/8	4/8	4/8
CPU Base Freq (GHz)		2.60	2.70	2.90
Intel® Turbo Boost Technology 2.0 Max Frequency (GHz)	**SC**	3.60	3.70	3.80
	DC	3.50	3.60	3.70
	QC	3.40	3.50	3.60
DDR3/DDR3L (MHz)		1600MHz	1600MHz	1600MHz
L3 Cache		6MB	8MB	8MB
Intel® HD Graphics 4000		Yes	Yes	Yes
Gfx Base Render Frequency		650MHz	650MHz	650MHz
Graphics Max Dynamic Frequency		1250MHz	1250MHz	1300MHz
PCIe Generation 3 Support		Yes	Yes	Yes
Intel® Secure Key		Yes	Yes	Yes
Intel® OS Guard		Yes	Yes	Yes
Intel® AES/TXT/vPro Technology		Yes	Yes	Yes
Intel® Virtualization Technology		Yes	Yes	Yes
Package		rPGA/ BGA-1224*	rPGA/ BGA-1224*	rPGA

- Was ist ein Prozessor?
- Wie ist ein einfacher Prozessor aufgebaut?
  - Wie ist ein Rechenwerk aufgebaut?
- Wie kann die Speicheranbindung klassifiziert werden?
  - Ist der Bus unidirektional?
  - Was ist der Unterschied zwischen primären und sekundären Speicher?

Inhalt

2.2 Flynn’sche Taxonomie (I)

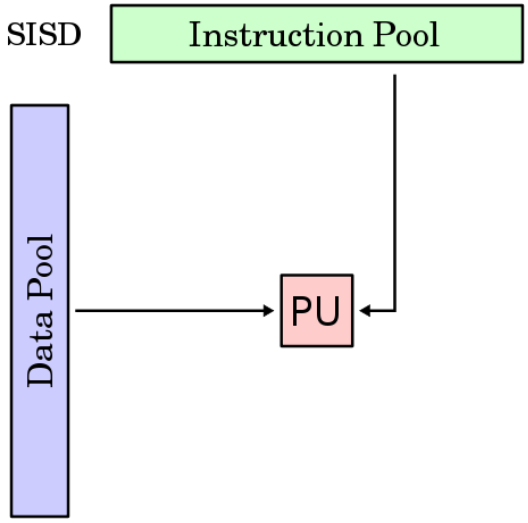
- Duale Unterscheidungsmerkmale:
  - Wie viele Instruktionen werden gleichzeitig ausgeführt?
    - Anzahl der Instruktionsströme (Hauptspeicher Prozessor)
  - Wie viele Datenworte verarbeitet eine Instruktion?
    - Anzahl der Datenströme (Hauptspeicher Prozessor Hauptspeicher)
  - Je Datenstrom und Instruktionsstrom zwei Klassen:
    - eins (single) oder viele (multiple)

Flynn’sche Klassifikation	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

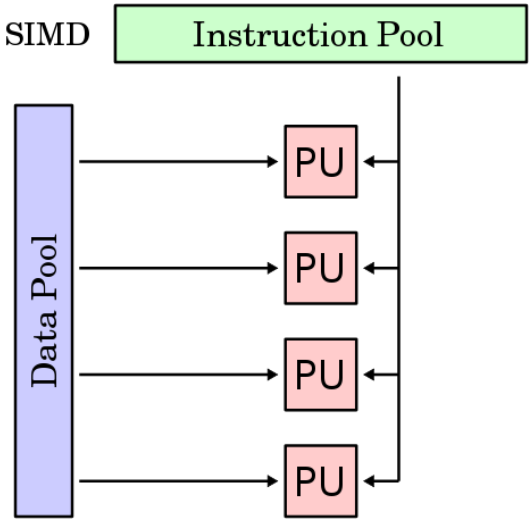
Flynn, Michael J.; , "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, vol.C-21, no.9, pp.948-960, Sept. 1972

2.2 Flynn’sche Taxonomie (II)

- Single Instruction, Single Data (SISD)
  - Von-Neumann-Architektur
    - gemeinsamer Daten- und Instruktions-Speicher
  - Harvard-Architektur
    - getrennter Daten- und Instruktions-Speicher

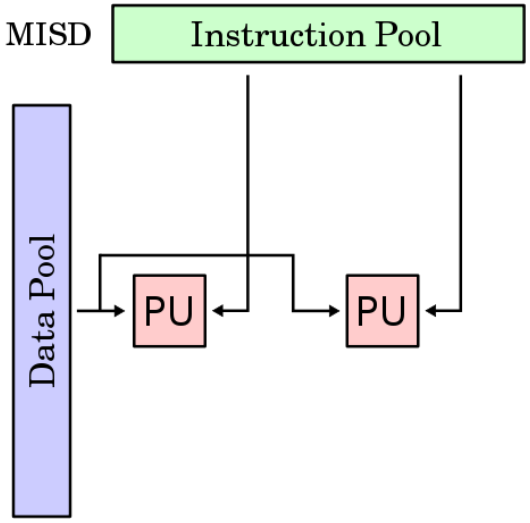


- Single Instruction, Multiple Data (SIMD)
  - Vektorprozessor
    - schnelle Ausführung gleichartiger Rechenoperationen auf mehrere gleichzeitig eintreffende oder zur Verfügung stehende Eingangsdatenströme

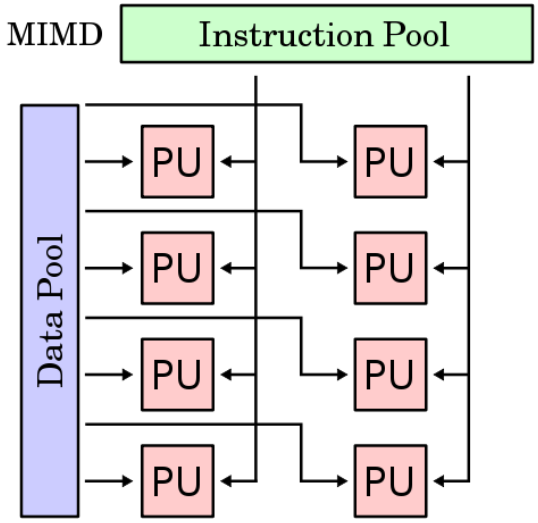


2.2 Flynn’sche Taxonomie (III)

- Multiple Instruction, Single Data (MISD)
  - Fehlertolerante, redundante Rechner
    - Redundante Datenströme zur Fehlererkennung bzw. –korrektur



- Multiple Instruction, Multiple Data (MIMD)
  - Multiprozessorsysteme
    - Führen gleichzeitig verschiedene Operationen auf verschieden gearteten Eingangsdatenströmen durch
    - Nebenläufige Kontrollflüsse
    - Kopplung über Verbindungs-Netzwerke



- Wie können Prozessoren klassifiziert werden?
- Welche Beispiele gibt es für die jeweiligen Klassen?
- Lassen sich alle heutigen Prozessoren eindeutig klassifizieren?

Inhalt

2.3 General Purpose Prozessor (GPP) (I)

- Eigenschaften
  - hohe Performanz für großes Anwendungsspektrum, nicht für eine spezielle Anwendung optimiert → universell
  - große Leistungsaufnahme, da i.a. hohe Taktraten
- Entwicklung
  - hochoptimierte Schaltungsstrukturen, Full-Custom + Standardzellen
  - Entwicklungszeit ~ 100 Personenjahre
  - nur in sehr großen Stückzahlen rentabel
- Einsatzgebiete
  - PCs, Workstations
  - Laptops

2.3.1 GPP: Architekturen

- Klassifizierung externer Architekturen
  - Wie werden Operanden von Befehlen im Prozessor gespeichert
  - Anzahl der Operanden pro Befehl
  - Residenz von Operanden
  - Operationsvorrat
  - Typ und Länge der Operanden

- Beispielklassen:
  - Akkumulator-Maschine
  - Stack-Maschine
  - Registersatzmaschine
    - RISC-Architektur
    - CISC-Architektur

2.3.1.1 GPP-Arch: Akkumulatormaschine (I)

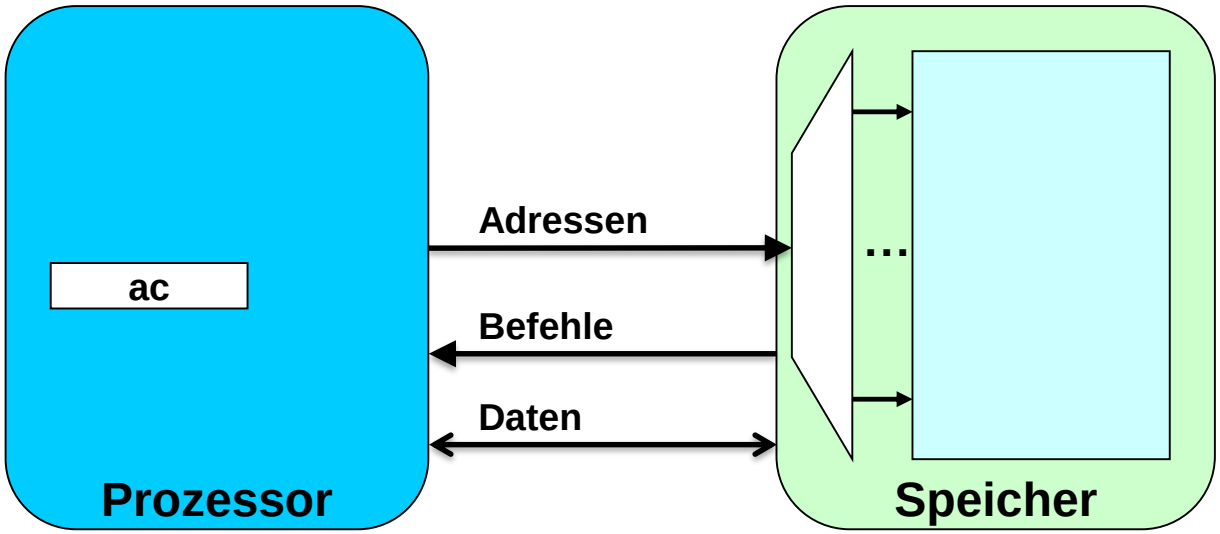
- Organisationsprinzip vieler einfacher, früherer Rechner, aber auch z.B. Motorola 6809

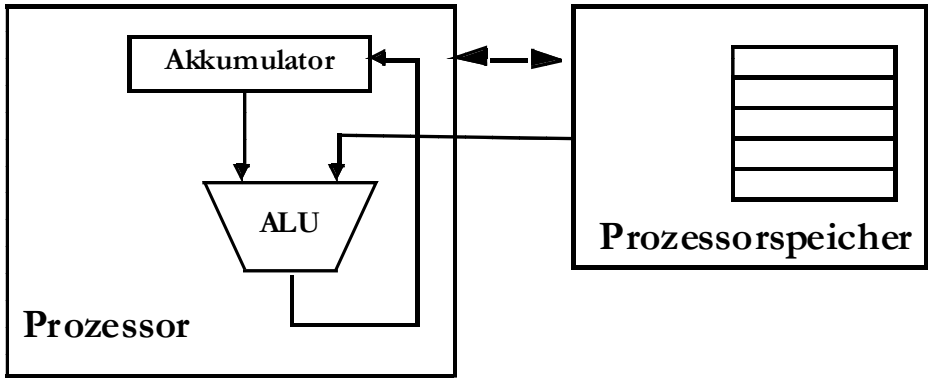
- Die Rechenergebnisse wurden in einem Register, dem Akkumulator (AC), „akkumuliert“
  - LAC m Lade Wert unter Adresse m in den AC
  - SAC m Speichere den Inhalt von AC nach Adresse m
  - ADD m Addiere den Inhalt von AC mit dem Wert unter Adresse m und speichere das Resultat nach AC

- Beispiel: c := a + b
  - LAC a
  - ADD b
  - SAC c

2.3.1.1 GPP-Arch: Akkumulatormaschine (II)

- Struktur:





2.3.1.2 GPP-Arch: Stackmaschine (I)

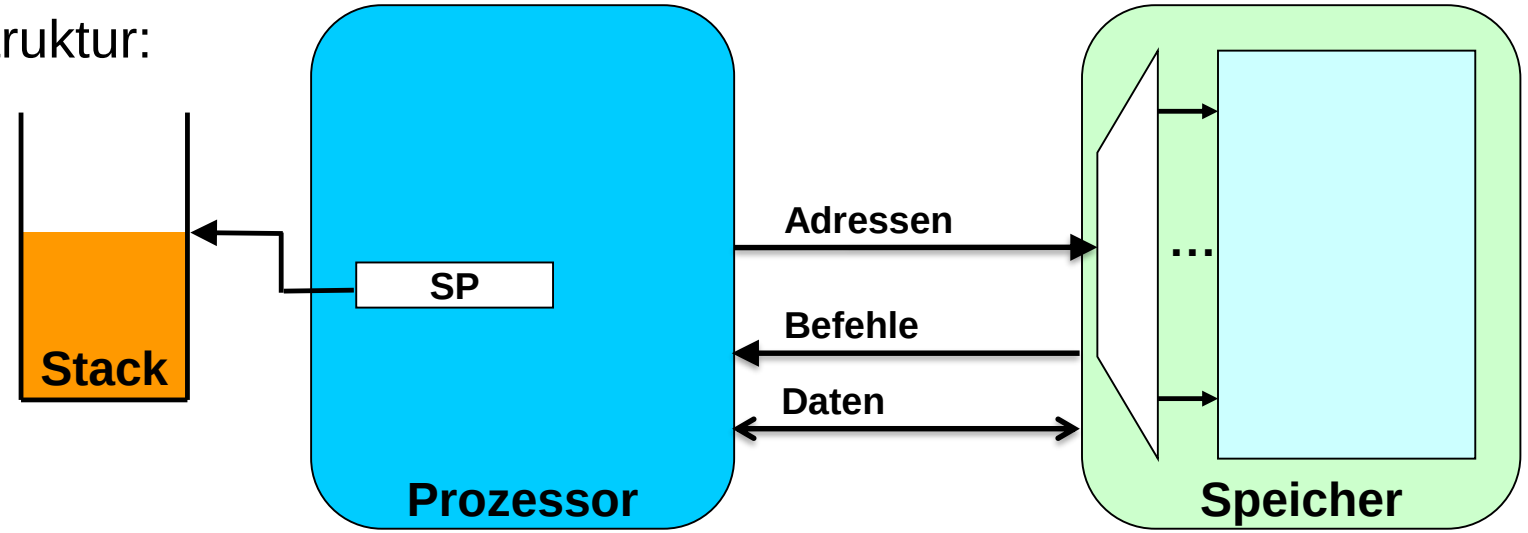
- Befehle beziehen sich auf einen Stack zur Speicherung u.a. von Zwischenergebnissen

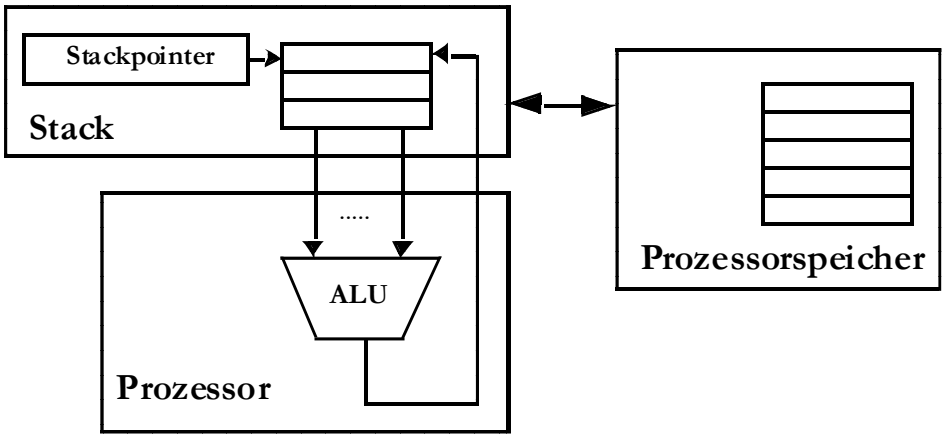
- Typische Befehle:
  - PUSH m Lege den Wert unter Adresse m auf den Stack
  - ADD Addiere die beiden obersten Werte des Stacks, entferne sie und lege die Summe als oberstes Element auf den Stack
  - STORE m Speichere das oberste Element auf dem Stack nach der Adresse m

- Beispiel: c := a + b
  - PUSH a
  - PUSH b
  - ADD
  - STORE c

2.3.1.2 GPP-Arch: Stackmaschine (II)

- Struktur:





2.3.1.3 GPP-Arch: Registersatzmaschine (I)

- Der Prozessor enthält z.B. 32 Universalregister

- Typische Befehle:
  - Datenverarbeitungs-Instruktionen
    - Arithmetic, Logisch, Compare, Move
  - Verzweigungs-Instruktionen
    - Branch
  - Register-Daten-Transfer-Instruktionen
    - Load, Store

- Üblich sind Dreiadressbefehle oder Zweiadressbefehle
  - OP DEST, SRC1, SRC2 DEST := SRC1 OP SRC2
  - OP DEST, SRC DEST := DEST OP SRC
    - Beispiel: ADD R5, R6 bedeutet R5 := R5 + R6

2.3.1.3 GPP-Arch: Registersatzmaschine (II)

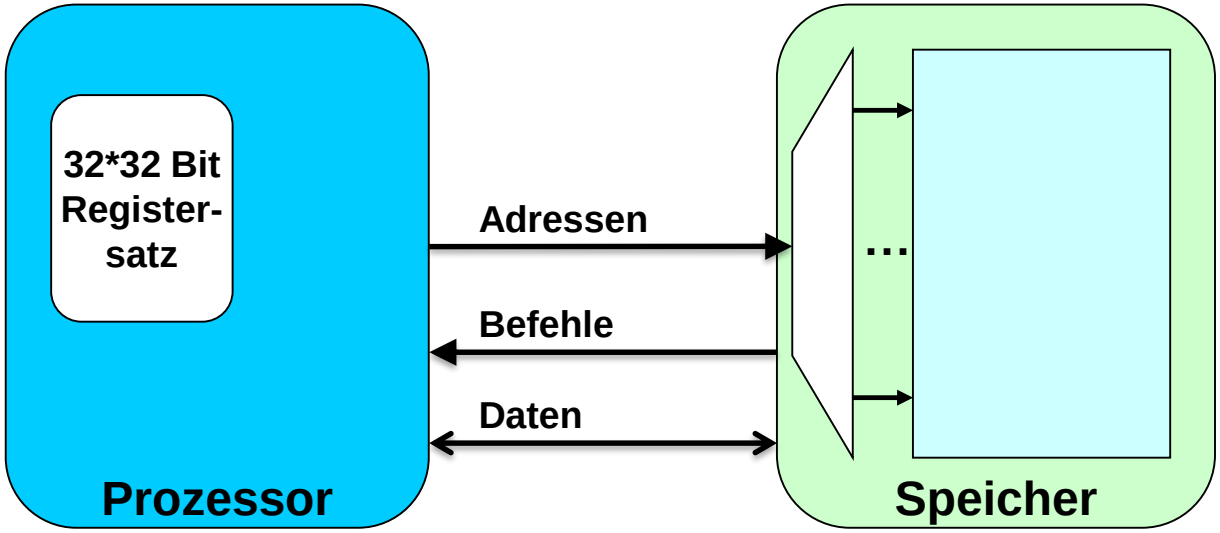
- Complex instruction set computer
  - Mehrere Taktzyklen pro komplexer Instruktion
  - Memory-to-Memory-Architektur
    - Load/Store sind eingebaute Instruktionen
  - Kleine Codegrößen
  - Benötigt mehr Transistoren zum Speichern komplexer Instruktionen
  - Schwerpunkt auf Hardware
  - Beispiel: c := a + b
    - MOVE R1, a
    - ADD R1, b
    - MOVE c, R1

- Reduced instruction set computer
  - Nur ein Taktzyklus pro reduzierter Instruktion
  - Load-Store-Architektur
    - Load-Store sind unabhängige Instruktionen
  - Große Codegrößen
  - Benötigt mehr Transistoren für Speicherregister
  - Schwerpunkt auf Software
  - Beispiel: c := a + b
    - LOAD R1, a
    - LOAD R2, b
    - ADD R3, R2, R1
    - STORE c, R3

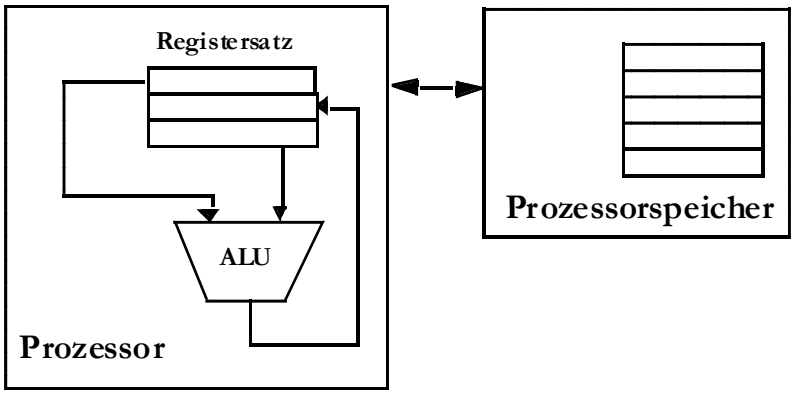
http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/risc/risccisc/

2.3.1.3 GPP-Arch: Registersatzmaschine (III)

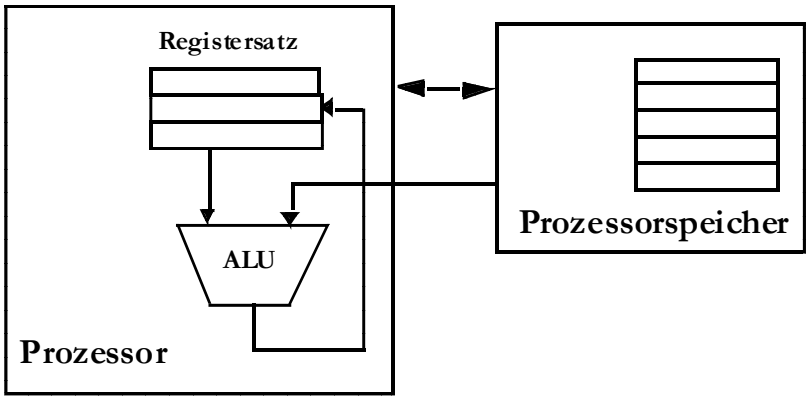
- Struktur:



- RISC



- CISC



2.3.1.3 GPP-Arch: Registersatzmaschine (IV)

- RISC vs. CISC
  - Programmgröße, transportierte Daten und Code sowie Ausführungszeiten:

$$Zeit/_{Programm} = Zeit/_{Zyklus} * Zyklen/_{Instruktion} * Instruktionen/_{Programm}$$

	Programmgröße:		**Transportierte Instr. und Daten:**		**CPU Zeit:**	
	VAX	**DEC 3100**	**VAX**	**DEC 3100**	**VAX**	**DEC 3100**
Gnu C Compiler	410kB	688kB	18MB	21MB	291s	90s
TeX	159kB	217kB	67MB	78MB	449s	95s
Spice	223kB	372kB	99MB	106MB	352s	94s

- RISC-Programme (DEC) dagegen sind länger bei gleichzeitig schnellerer Ausführung
- CISC-Programme (VAX) sind meist kleiner, benötigen jedoch eine längere Ausführungszeit

- Wie können General Purpose Prozessoren unterschieden werden?
- Wovon hängt die Geschwindigkeit eines GPP ab?
- Was ist der Nachteil eines RISC-Prozessors?
- Was ist ein Compiler?

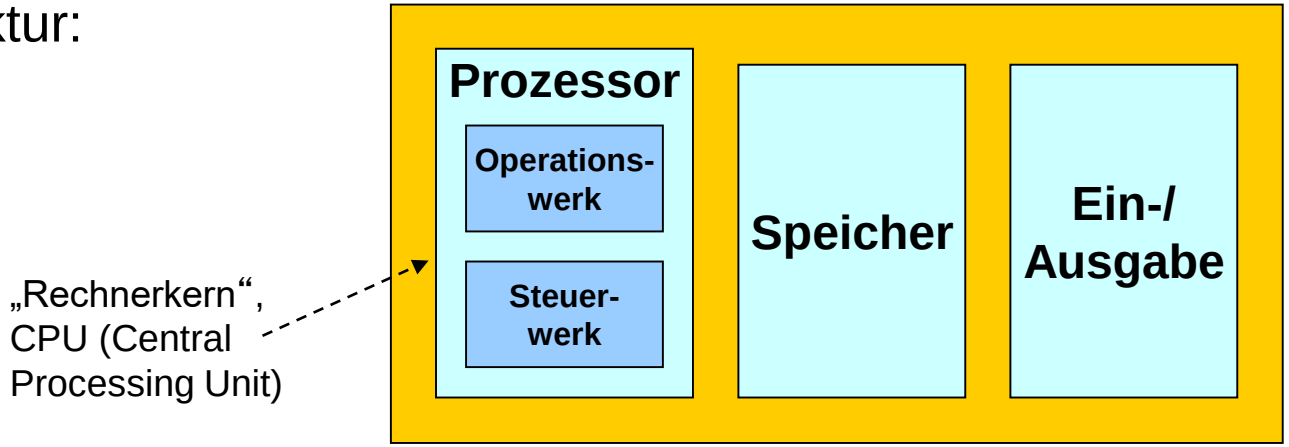
Inhalt

2.3.2 GPP: Performanzsteigerung (I)

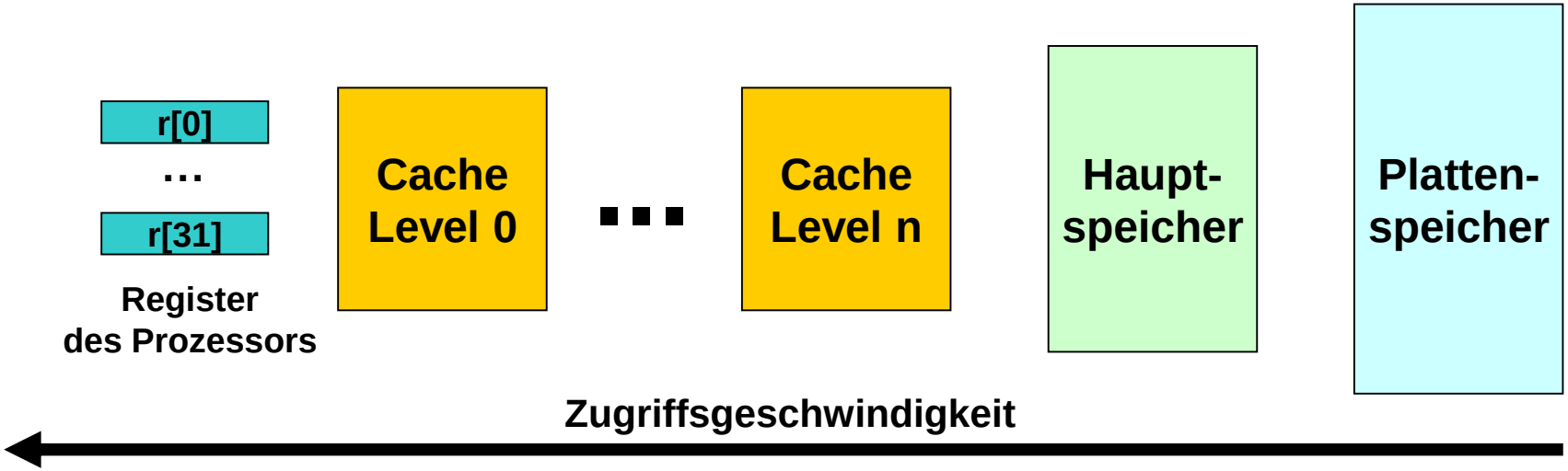
- Pipeline und tiefe Instruktions-Pipeline
  - z.B.: fetch | decode | read | execute | write back
  - Sprungvorhersage
- Mehrere skalare Einheiten
  - z.B.: Integer Units, Floating-Point Unit, Load/Store unit
  - Je nach skalarer Einheit verschieden tiefe Pipelines
- Dynamic Scheduling und Out-of-Order Execution
  - Prozessor erkennt parallel ausführbare Instruktionen und weist sie den Einheiten zu.
  - komplexes Steuerwerk
- VLIW
  - Compiler erkennt Parallelität und plant die Abfolge der Instruktionen auf den verschiedenen Funktionseinheiten.

2.3.2 GPP: Performanzsteigerung (II)

- Interne Architektur:



- Speicherhierarchie in einem Rechner:



Inhalt

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 GPP: Pipelining (II)

- Unter dem Begriff **Pipelining** versteht man die Zerlegung einer Maschinenoperation in mehrere Phasen, die dann von hintereinander geschalteten Verarbeitungseinheiten taktsynchron bearbeitet werden, wobei jede Verarbeitungseinheit genau eine spezielle Teiloperation ausführt.

- Die Gesamtheit dieser Verarbeitungseinheiten nennt man eine **Pipeline**.

- Bei einer **Befehlspipeline / Instruction Pipeline** wird die Ausführung eines Maschinenbefehls in verschiedene Phasen unterteilt, aufeinanderfolgende Maschinenbefehle werden jeweils um einen Taktzyklus versetzt ausgeführt.

2.3.2.1 GPP: Pipelining (III)

- Jede Stufe der Pipeline heißt **Pipeline-Stufe** oder **Pipeline-Segment**.
- Pipeline-Stufen werden durch getaktete **Pipeline-Register** (auch latches genannt) getrennt.
- Ein **Pipeline-Maschinentakt** ist die Zeit, die benötigt wird, um einen Befehl eine Stufe weiter durch die Pipeline zu schieben.
- Idealerweise wird ein Befehl in einer k-stufigen Pipeline in k Takten von k Stufen ausgeführt.
- Wird in jedem Takt ein neuer Befehl geladen, dann werden zu jedem Zeitpunkt unter idealen Bedingungen k Befehle gleichzeitig behandelt und jeder Befehl benötigt k Takte, bis zum Verlassen der Pipeline.

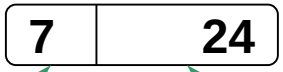
2.3.2.1 GPP: Pipelining (IV)

- Man definiert die **Latenz** als die Zeit, die ein Befehl benötigt, um alle k Pipeline-Stufen zu durchlaufen.

- Der **Durchsatz** einer Pipeline wird definiert als die Anzahl der Befehle, die eine Pipeline pro Takt verlassen können.
  Dieser Wert spiegelt die Rechenleistung einer Pipeline wieder.

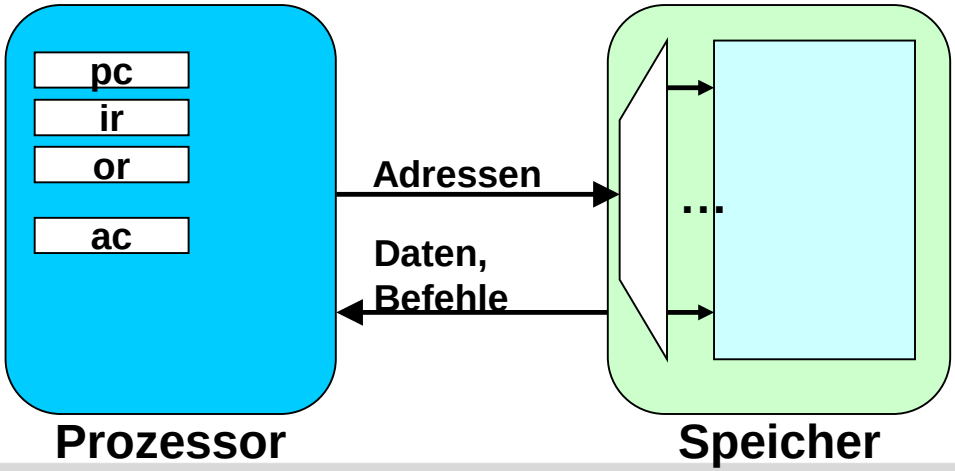
2.3.2.1 Exemplarische Befehlsausführung (I)

- Verarbeitungsphasen eines Befehls am Beispiel
  - Annahme: einfache Akkumulatormaschine
  - Befehlsformat



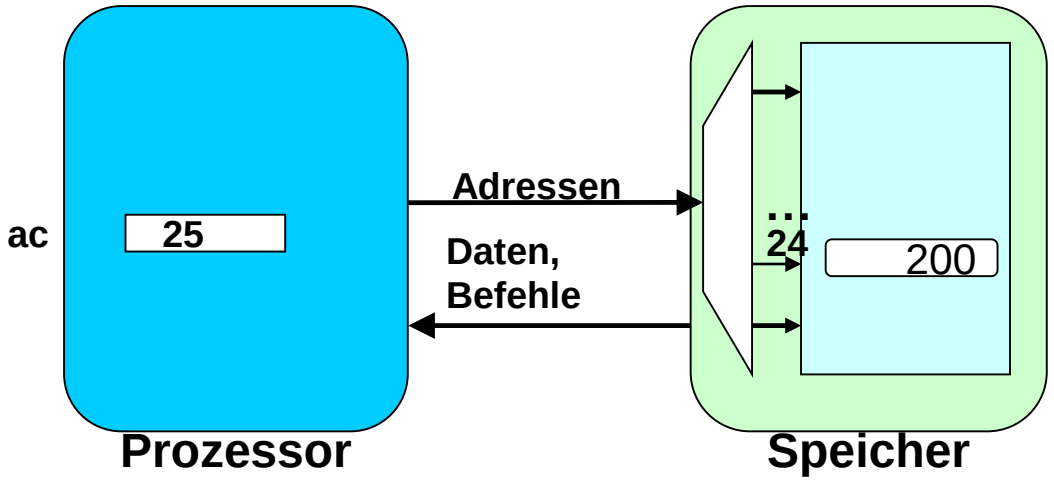
  - Befehlscode
  - Adresse 24
  - Addiere-Befehl ≅ 7

- Der Addiere-Befehl holt den Wert unter der im Befehl angegebenen Adresse (im Beispiel 24) und addiert ihn zum Inhalt

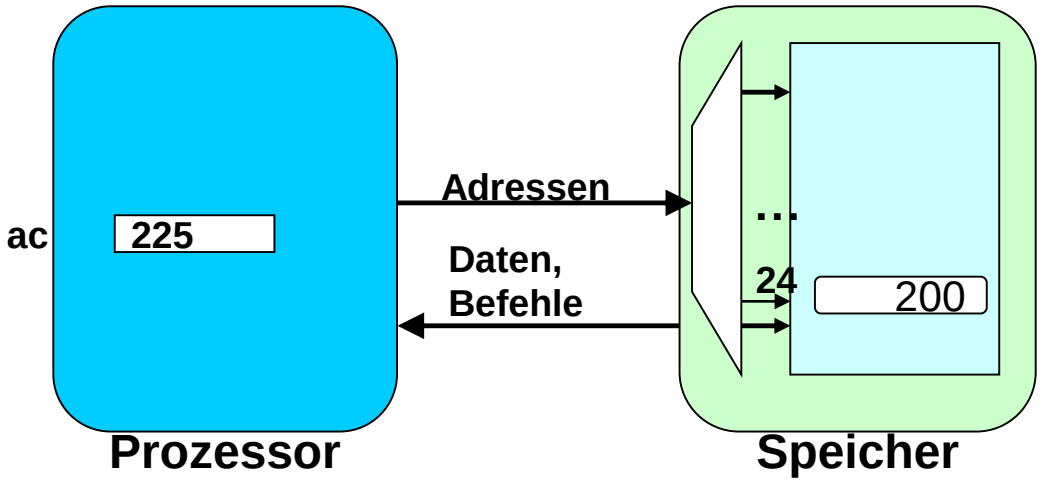


2.3.2.1 Exemplarische Befehlsausführung (II)

- Befehlscode: Addiere-Befehl ≅ 7 → | 7 | 24 | ← Adresse ≅ 7
- Vorher:

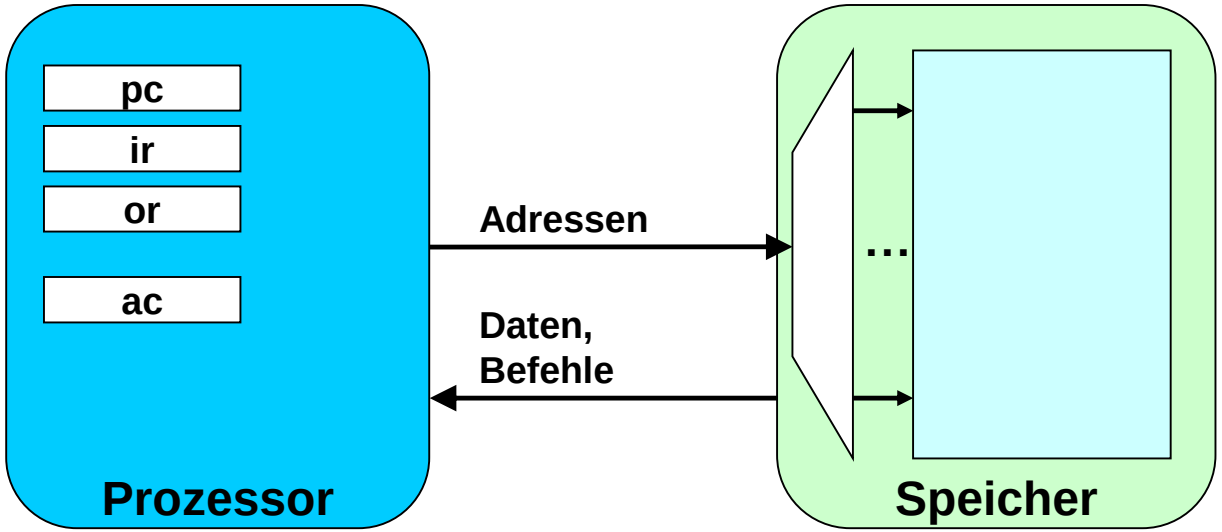


- Nachher:



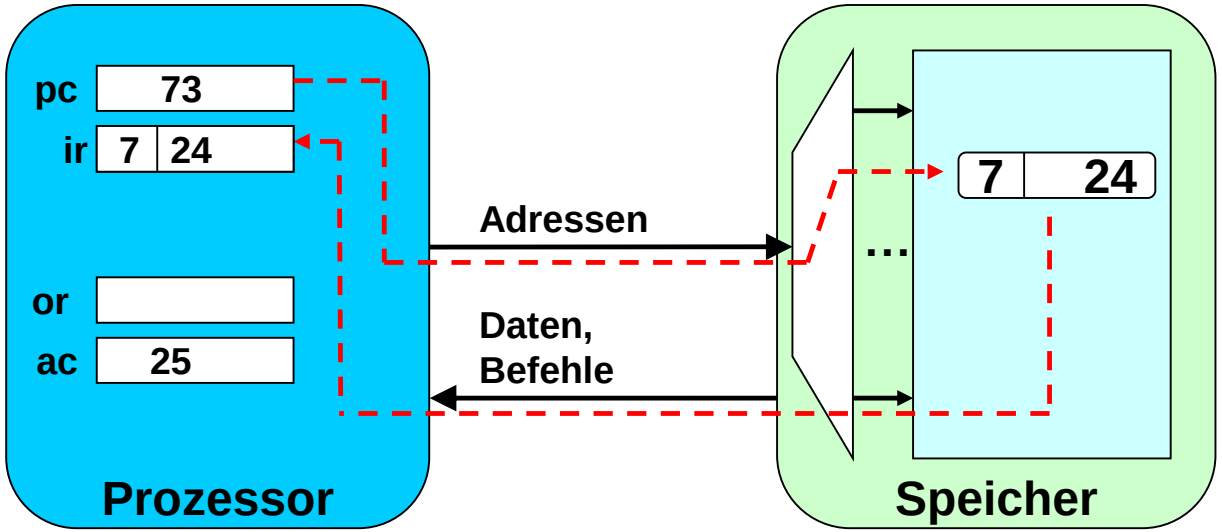
2.3.2.1 Exemplarische Befehlsausführung (III)

- Struktur:
  - pc: program counter, auch
  - ip: instruction pointer genannt
  - ir: instruction register
  - or: operand register
  - ac: accumulator



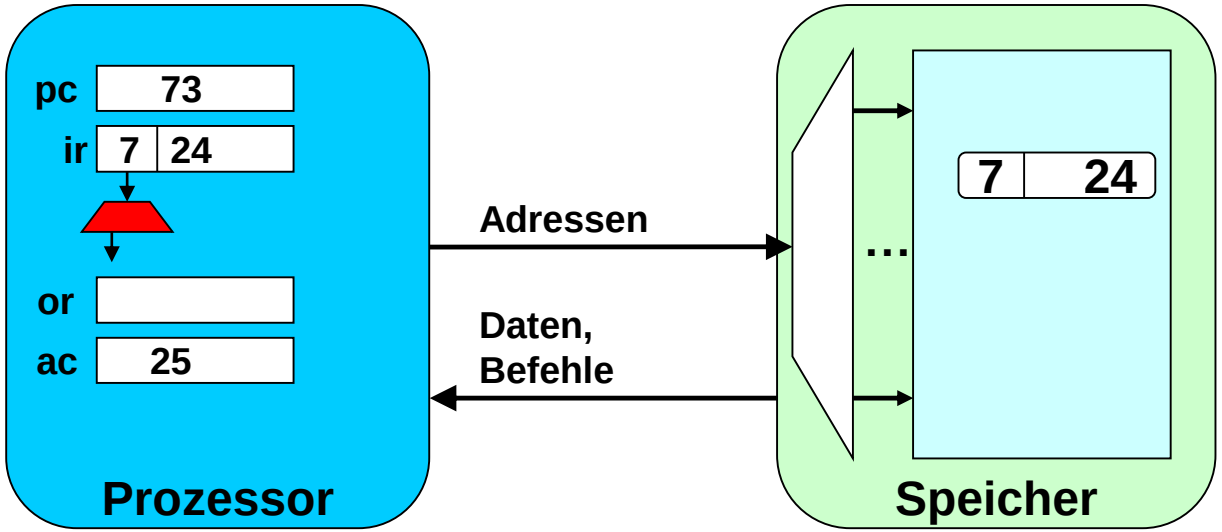
2.3.2.1 Exemplarische Befehlsausführung (IV)

- 1. Befehlsholphase (instruction fetch, IF)



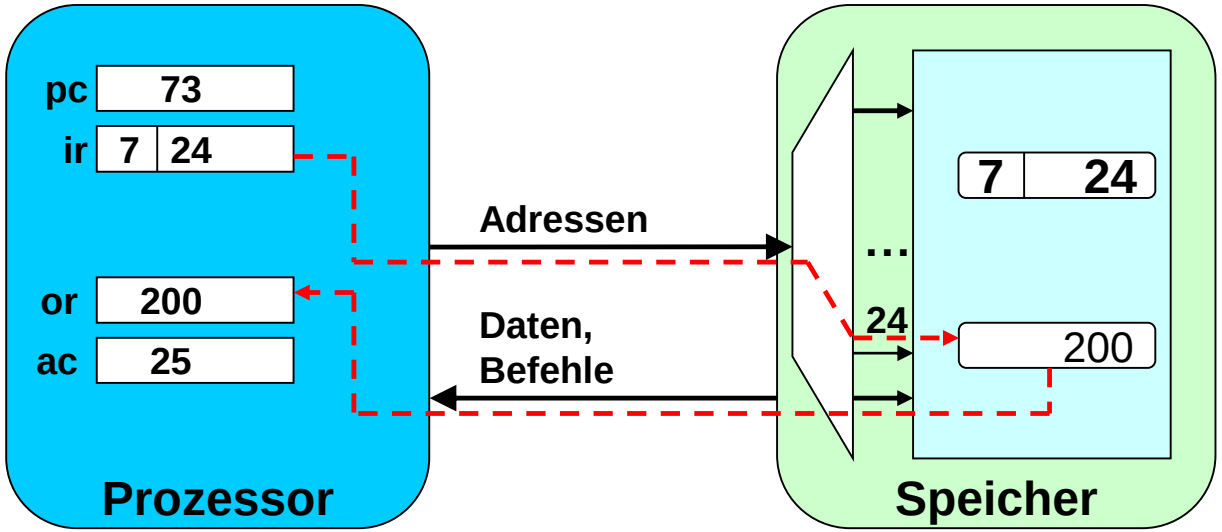
2.3.2.1 Exemplarische Befehlsausführung (V)

- 2. Befehl dekodieren (instruction decode, ID)



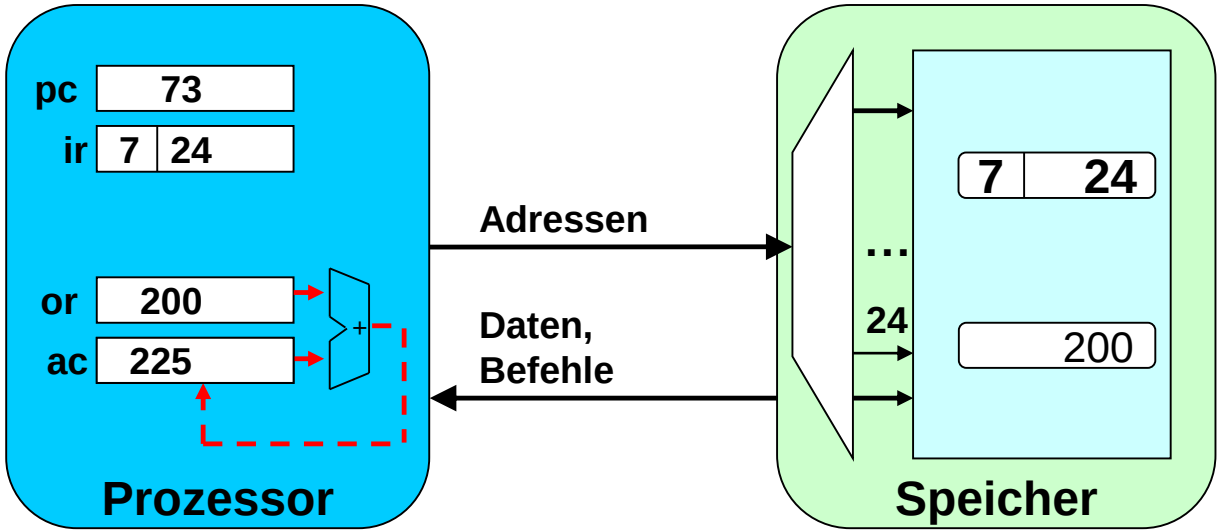
2.3.2.1 Exemplarische Befehlsausführung (VI)

- 3. Operand holen (operand fetch, OF)



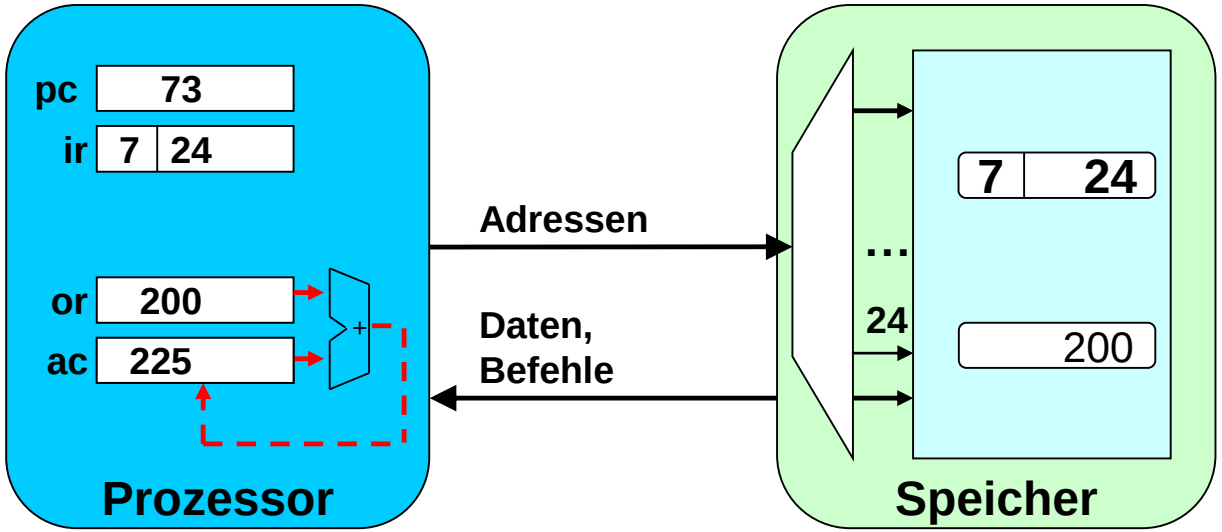
2.3.2.1 Exemplarische Befehlsausführung (VII)

- 4. Befehl ausführen (instruction execute, EX)



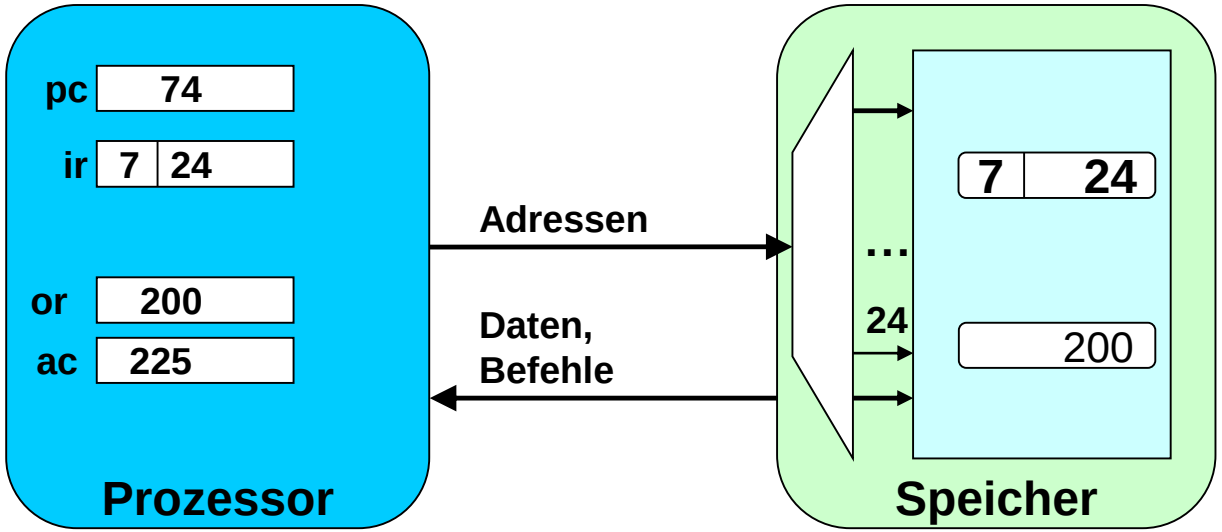
2.3.2.1 Exemplarische Befehlsausführung (VIII)

- Befehlszähler inkrementieren
  - kann als separater Schritt oder parallel erfolgen



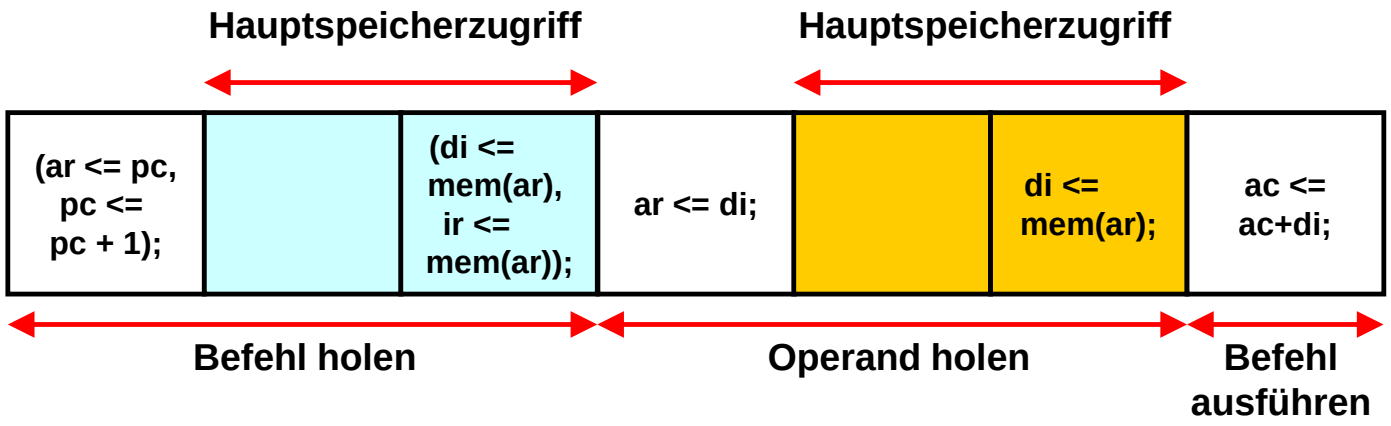
2.3.2.1 Exemplarische Befehlsausführung (IX)

- Resultat:
  - Ergebnis 225 steht im accumulator ac



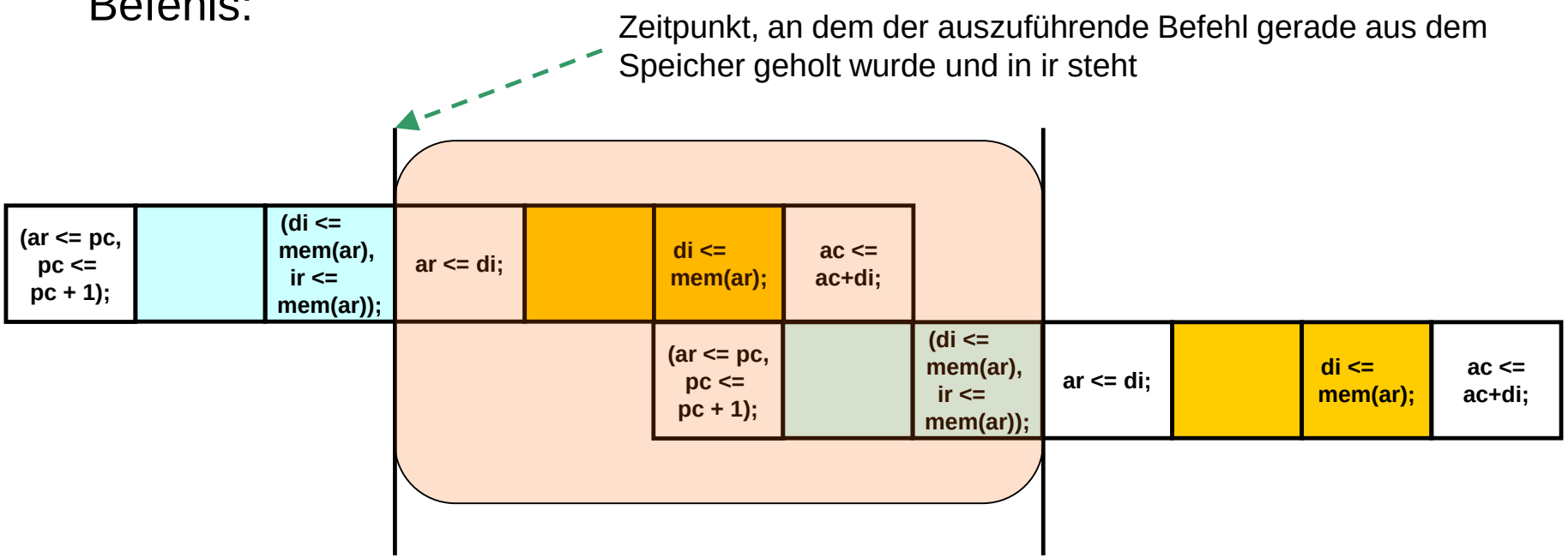
2.3.2.1 Exemplarische Befehlsausführung (X)

- Annahmen:
  - Hauptspeicherzugriffzeit = 2 * Prozessortaktzeit
  - Befehlsauslegung für größtmögliche Geschwindigkeit
  - Beispiel: Addiere-zu-Akkumulator-Befehl
  - Auslastung bei serieller Ausführung:
    - Hauptspeicher: 57%
    - CPU: 71%



2.3.2.1 Exemplarische Befehlsausführung (XI)

- Überlappung der Ausführungsphase mit dem Holen des nächsten Befehls:

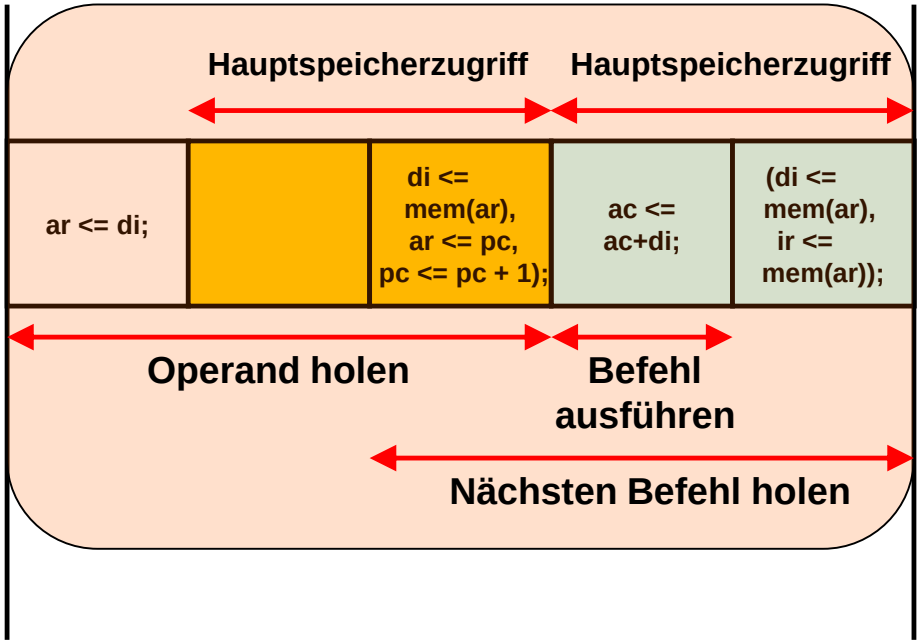


- Es werden keine zusätzlichen Ressourcen benötigt

- Eine zeitliche Einsparung von 2 Takten pro Befehl => 28,6 % Leistungssteigerung

2.3.2.1 Exemplarische Befehlsausführung (XII)

- Während eines Speicherzugriffs wird bereits der nächste Speicherzugriff vorbereitet
- Ebenfalls Parallel zum Speicherzugriff: Kalkulation des Ergebnisses
- Nur noch 5 Takte notwendig, keine zusätzlichen Kosten
- Auslastung bei überlappter Ausführung:
  - Hauptspeicher: 80%
  - CPU: 80%



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)

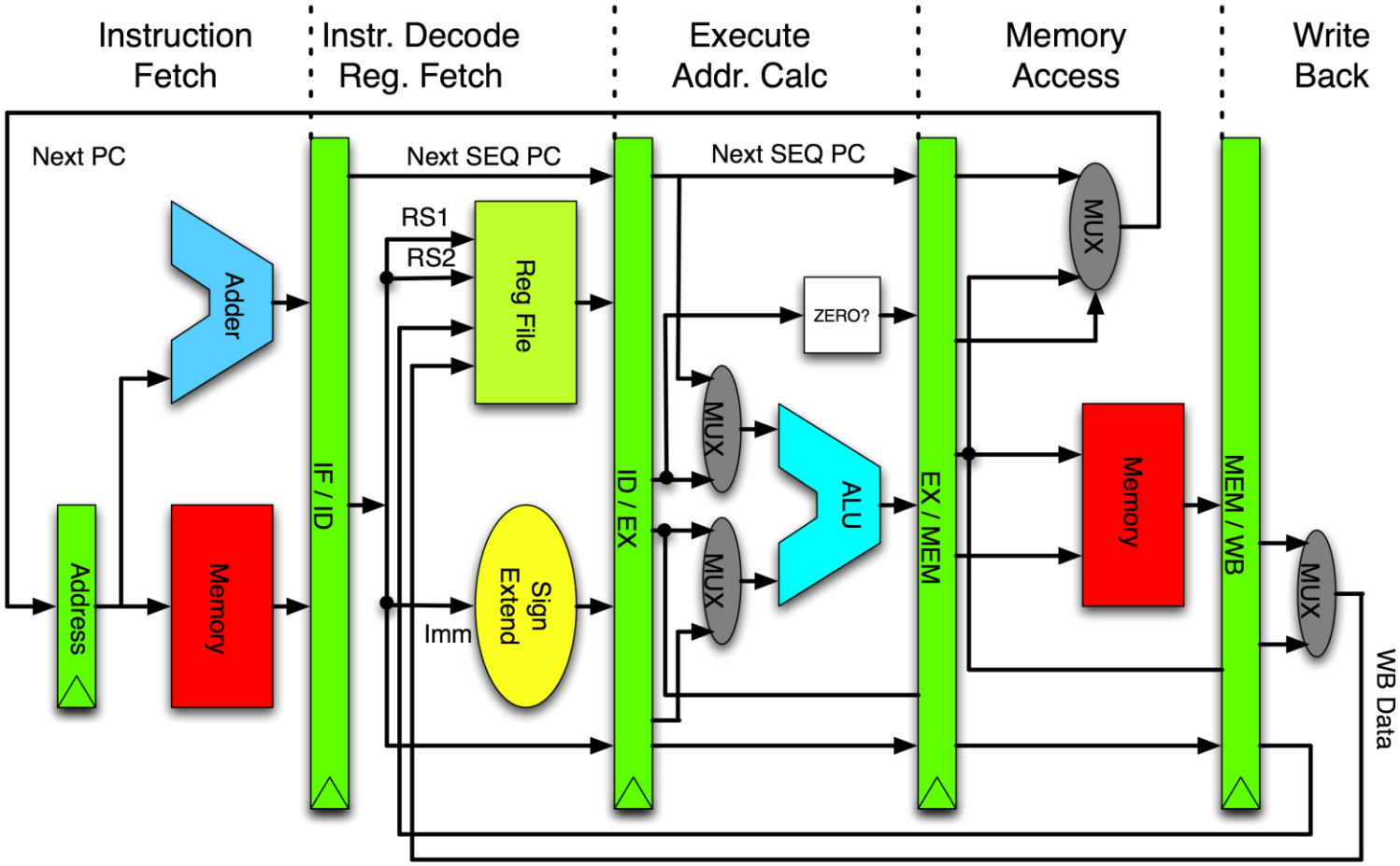
2.3.2.1 Phasen der Befehlsausführung einer 5-stufigen Befehlspipeline (II)

- 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) Anmerkung: z.B. Zugriff auf Cache oder Hauptspeicher, aber nicht (!) Register

2.3.2.1 RISC Prozessor 5 stufige pipeline

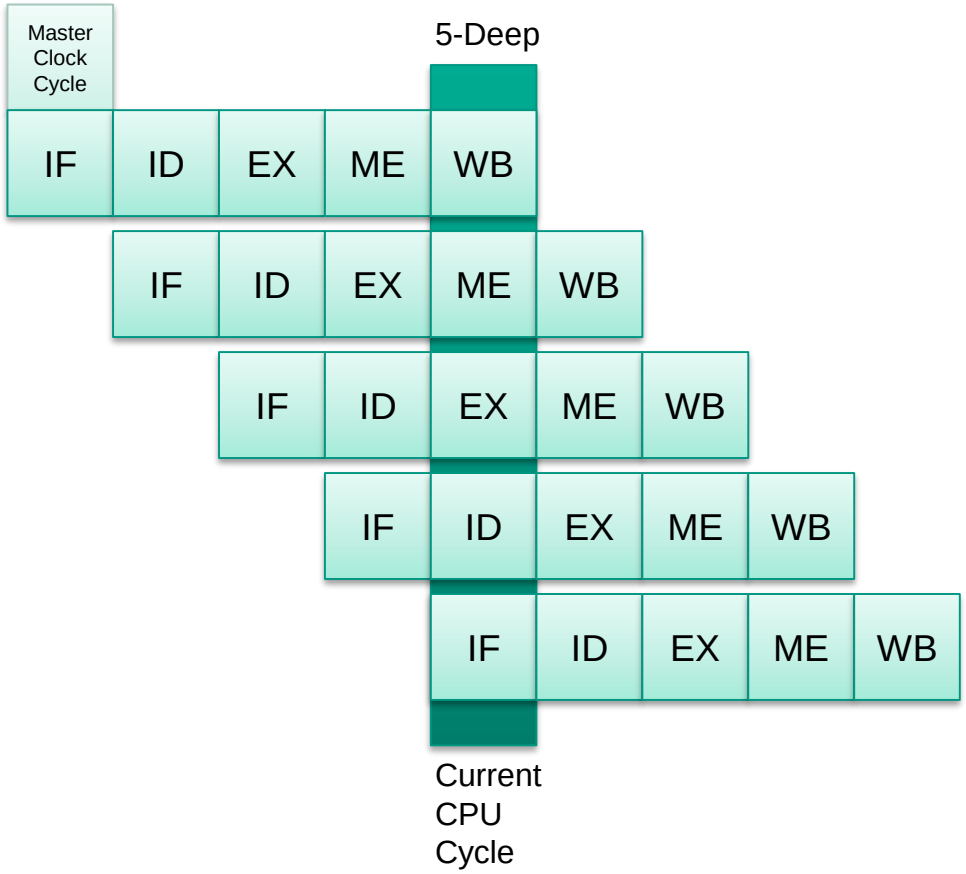
Entwickelt von MIPS Technologies (früher MIPS Computer Systems, Inc.).



Die stufenweise Abbildung der Pipeline des MIPS Mikroprozessord. Der Speicher ist für die Klarheit der Pipeline doppelt dargestellt. Es handelt sich um denselben Speicher (von Neumann Architektur)

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 Beispiel: Optimales Pipelining

- Cycle

1	2	3	4	5	6	7	8	9	10

- Operation

ADD	IF	ID	EX	ME	WB				
SUB		IF	ID	EX	ME	WB			
OR			IF	ID	EX	ME	WB		
AND				IF	ID	EX	ME	WB	
OR					IF	ID	EX	ME	WB
XOR						IF	ID	EX	ME	WB

- Alle Operationen arbeiten auf Registern (single cycle execution)
- Zu jedem Taktzyklus wird eine Instruktion beendet
  - Erinnerung: Latenz bleibt identisch aber Durchsatz wird erhöht (5x)

2.3.2.1 Pipeline Konflikte (I)

- Erreicht man immer den n-fachen Durchsatz bei einer n-stufigen Pipeline? → NEIN, da es zu Pipelinekonflikten kommen kann.

D.H.:

- Die nächste Instruktion im Befehlsstrom im zugewiesenen Taktzyklus wird nicht ausgeführt
- Unterbrechung des taktsynchronen Durchlaufs durch die einzelnen Stufen der Pipeline
- Verursacht Leistungseinbußen, da Durchsatz reduziert wird

2.3.2.1 Pipeline Konflikte (II)

- 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 (NOP)
  - 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

2.3.2.1 Beispiel: Load Pipeline

- Cycle

1	2	3	4	5	6	7	8	9	10	11	12

- Operation

ADD	IF	ID	EX	ME	WB						
SUB		IF	ID	EX	ME	WB					
LD			IF	ID	EX	ME	WB				
AND				IF	ID	S	S	EX	ME	WB	
OR					IF	S	S	ID	EX	ME	WB
XOR								IF	ID	EX	ME	WB

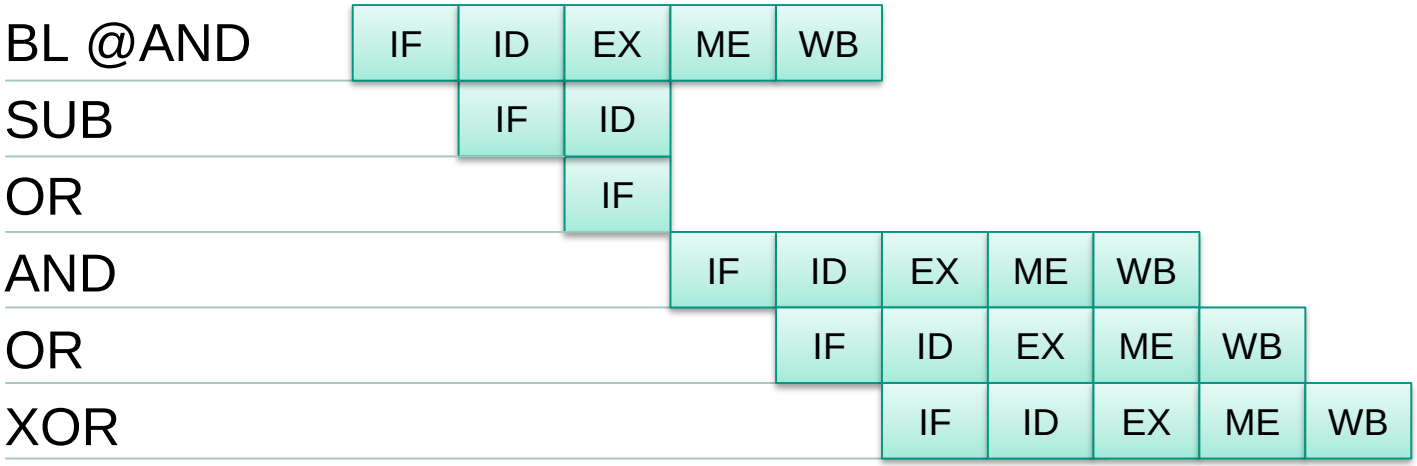
- Stalling für zwei Taktzyklen
- 6 Taktzyklen für 4 Instruktionen
  - Clock cycles per Instruction (CPI) = 1,5

S Stall

2.3.2.1 Beispiel: Branch and Link Pipeline

- Cycle

- Operation



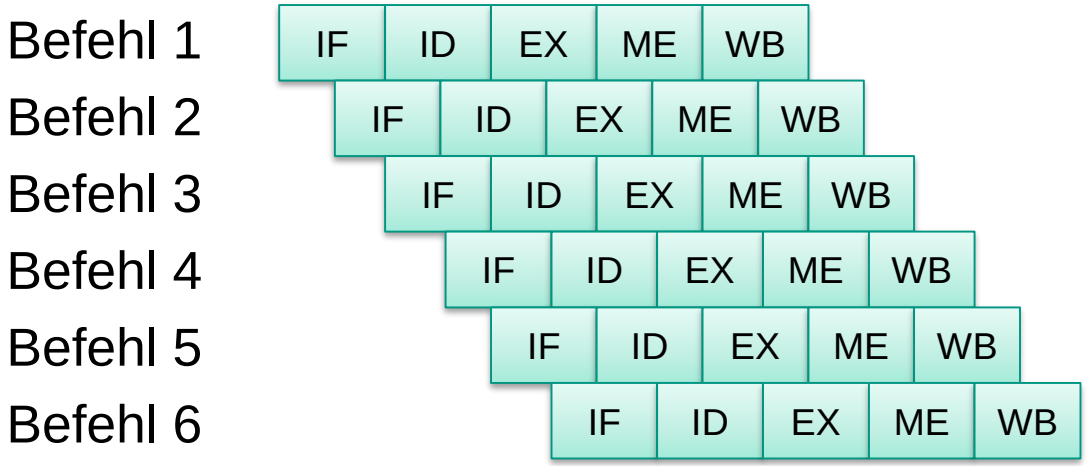
- Aufbrechen der Pipeline

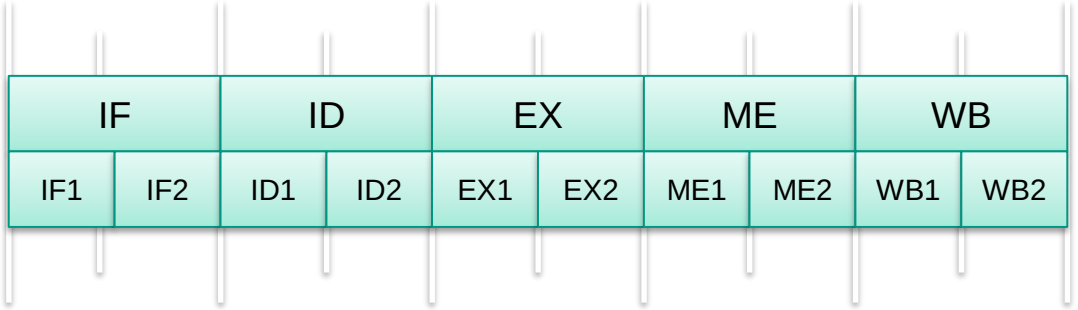
2.3.2.1 Beispiel: Pipeline - Datenkonflikte

- Datenkonflikte
  - Datenabhängigkeiten (zwischen Befehlen im Programm)
  - Beispiel:
    - ADD R1, R2, R3
    - AND R6, R1, R8
    - XOR R9, R1, R11

S Stall

2.3.2.1 Superpipelining (I)

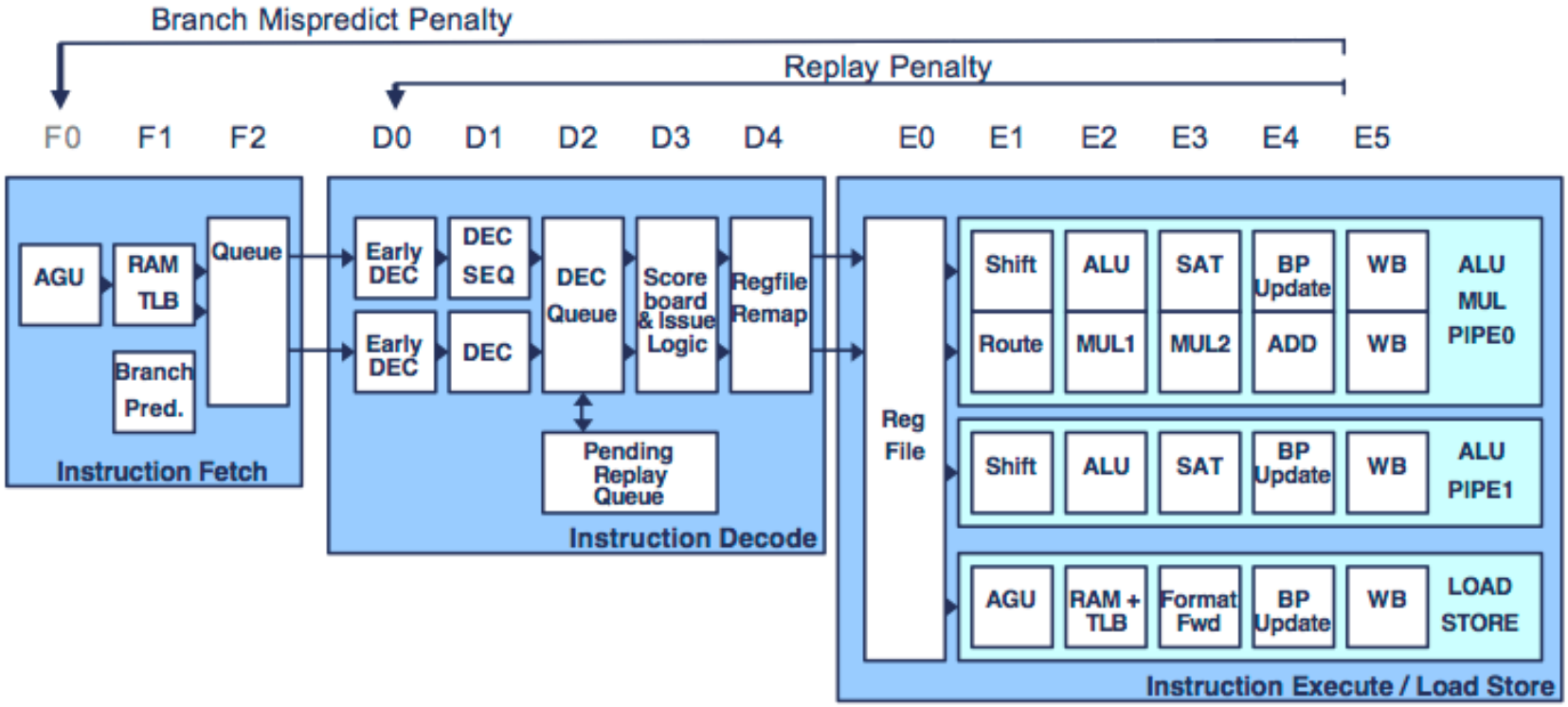




Maschinenzyklus
½ Grundtakt der
RISC-Maschine

2.3.2.1 Superpipelining (II)

- ARM Cortex A8 integer pipeline



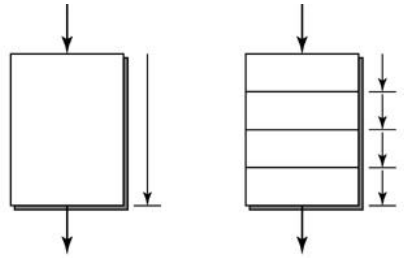
- Optimierung von Code zur Ausnutzung der Pipeline sehr schwierig
  - Compiler-Aufgabe!

- Was ist Pipelining?
- Wie ist die 5-stufige DLX Pipeline aufgebaut?
- Welche Latenz hat die Pipeline bei einem Takt pro Stufe?
- Was ist der ideale Durchsatz des Prozessors? Kann er erreicht werden?
- Welche Konflikte können auftreten?
- Wie können diese gelöst werden?
- Warum gibt es keine 50-stufigen Pipelines?

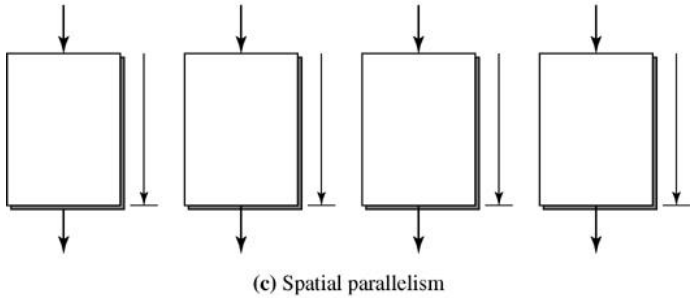
Inhalt

2.3.2.2 Superskalarität (I)

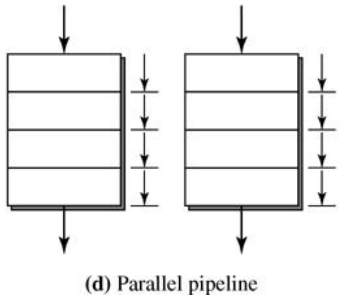
- a Keine Parallelität (Nonpipelined)
- b Zeitliche Parallelität (Pipelined)



- c Räumliche Parallelität (Multiple units)



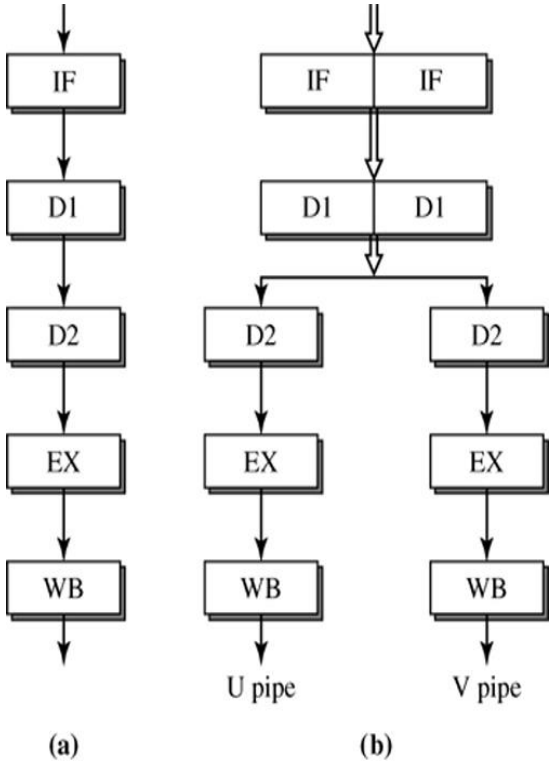
- d Zeitliche und Räumliche Parallelität



Parts from: Prof. Lizy Kurian John Univ. Austin, Texas

2.3.2.2 Superskalarität (II)

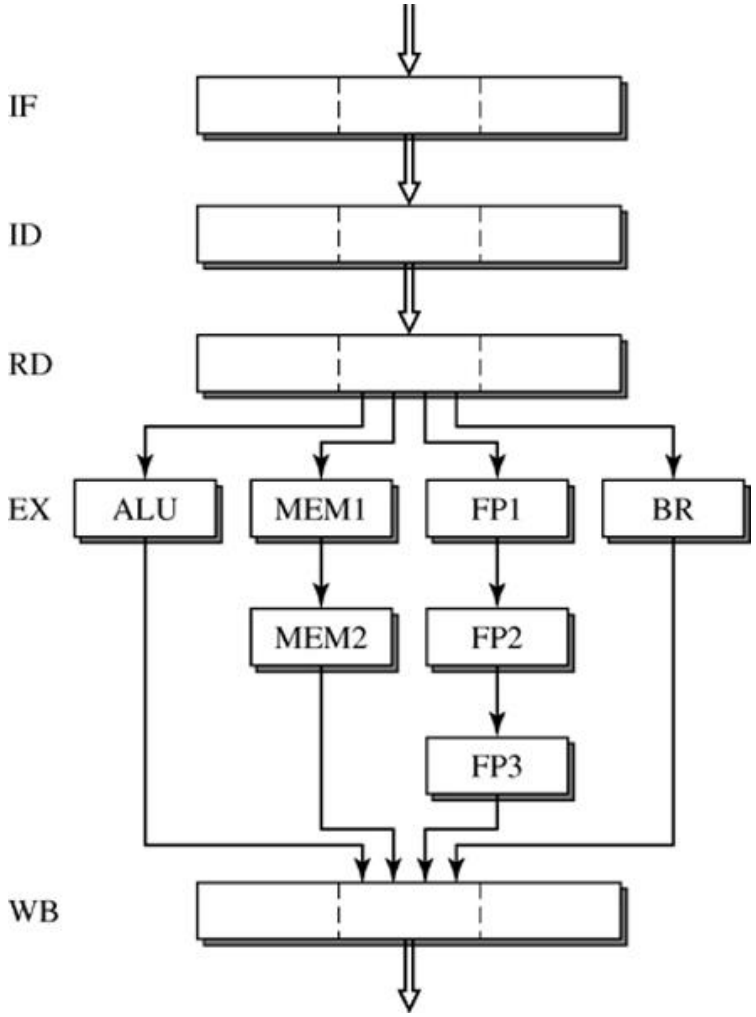
- Fetch
  - Load 16-bytes of instruction into prefetch buffer
- Decode1
  - Determine instruction length, instruction type
- Decode2
  - Compute memory address
  - Generate immediate operands
- Execute
  - Register Read
  - ALU operation
  - Memory read/write
- Write-Back
  - Update register file



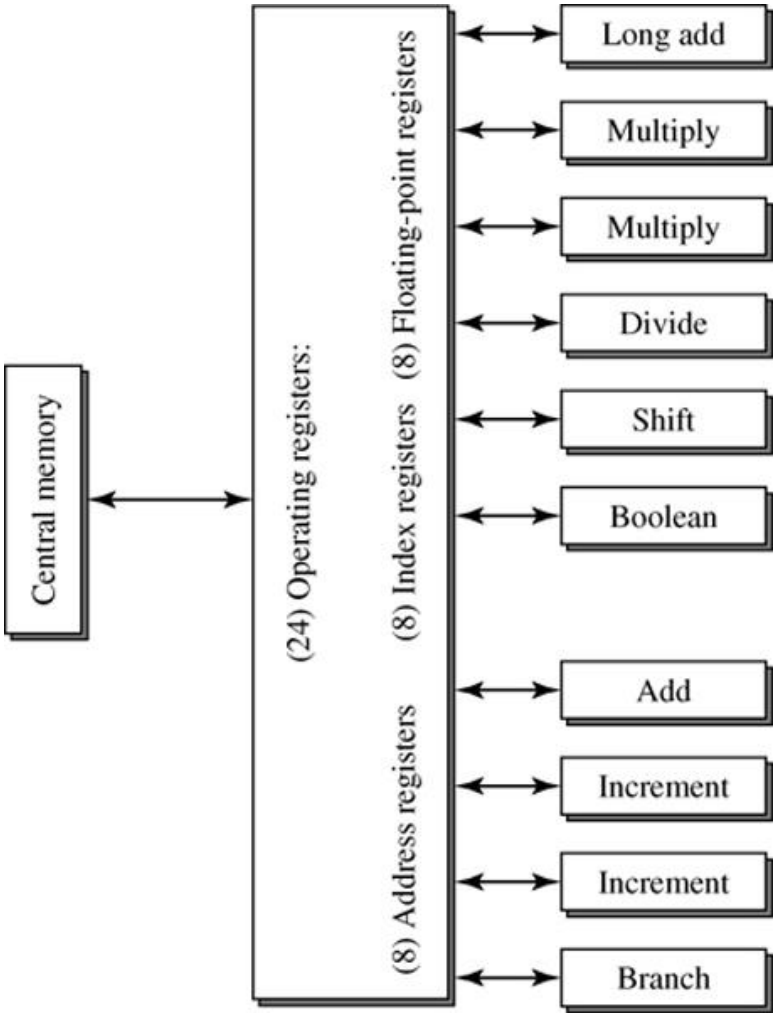
5-stufige i486 skalare Pipeline

5-stufige Pentium parallele Pipeline

2.3.2.2 Superskalarität (III)



Parts from: Prof. Lizy Kurian John Univ. Austin, Texas



CDC 6600 mit 10 verschiedenartigen funktionalen Einheiten

2.3.2.2 Superskalarität – Begiffe (I)

- Superpipeline
  - Tiefe Pipeline, aber keine superskalaren Einheiten
- Superskalar
  - Kann mehr als eine Instruktion/Zyklus ausführen
- Out-of-Order
  - Kann Instruktionen außerhalb der Programmreihenfolge ausführen
- Spekulation
  - Führt Instruktionen auch nach Branches weiter aus, die später eventuell annulliert werden müssen

2.3.2.2 Superskalarität – Begiffe (II)

- Mehrere Funktionseinheiten, parallel arbeitend
- Den Ausführungseinheiten kann mehr als ein Befehl pro Takt zugewiesen werden
- Die Befehle werden aus einem sequenziellen Strom von normalen Befehlen zugewiesen
- Die Zuweisung der Befehle erfolgt in Hardware durch einen dynamischen Scheduler (-> Dispatcher)
- Die Anzahl der zugewiesenen Befehle pro Takt wird dynamisch von der Hardware bestimmt und liegt zwischen null und der maximal möglichen Zuweisungsbandbreite

2.3.2.2 Superskalarität – Begiffe (III)

- Die dynamische Zuweisung von Befehlen führt zu einem komplexen Hardware-Scheduler.
  - Die Komplexität des Schedulers steigt mit der Größe des Befehlsfensters und mit der Anzahl der Befehle, die außerhalb der Programmreihenfolge zugewiesen werden können.

- Mehrere Ausführungseinheiten müssen verfügbar sein
  - Anzahl der Ausführungseinheiten entspricht mindestens der Zuweisungsbandbreite,
  - Häufig gibt es jedoch noch mehr Ausführungseinheiten, um potenzielle Strukturkonflikte zu umgehen.

2.3.2.2 Superskalarität – Begiffe (IV)

- Superskalartechnik ist eine Mikroarchitekturtechnik und hat keinen Einfluss auf die Befehlssatz-Architektur.

- Der Begriff „superskalar“ wird oft in unpräziser Weise benutzt, um Prozessoren mit mehreren parallelen Pipelines oder mehreren Ausführungseinheiten zu beschreiben. Gilt jedoch nur dann, wenn die Zuweisung der Instruktionen dynamisch erfolgt
  - In dieser Weise kann man nicht zwischen der Superskalar- und der VLIW-Technik zu unterscheiden.

2.3.2.2 Superskalarität – Begiffe (V)

- Befehls-Pipelining und Superskalartechnik nutzen beide feinkörnige Parallelität (fine-grain oder instruction-level parallelism), d.h. Parallelität zwischen einzelnen Befehlen.
  - Pipelining nutzt zeitliche Parallelität (temporal parallelism)
  - Superskalar nutzt räumliche Parallelität (spatial parallelism).

- Eine Leistungssteigerung durch zeitliche Parallelität kann mit einer längeren Pipeline und höherer Taktfrequenz erreicht werden.

- Falls genügend feinkörnige Parallelität vorhanden ist, kann die Leistung durch räumliche Parallelität im superskalaren Fall mit Hilfe von mehr Ausführungseinheiten und einer hohen dynamischen Zuweisungsrate erreicht werden.

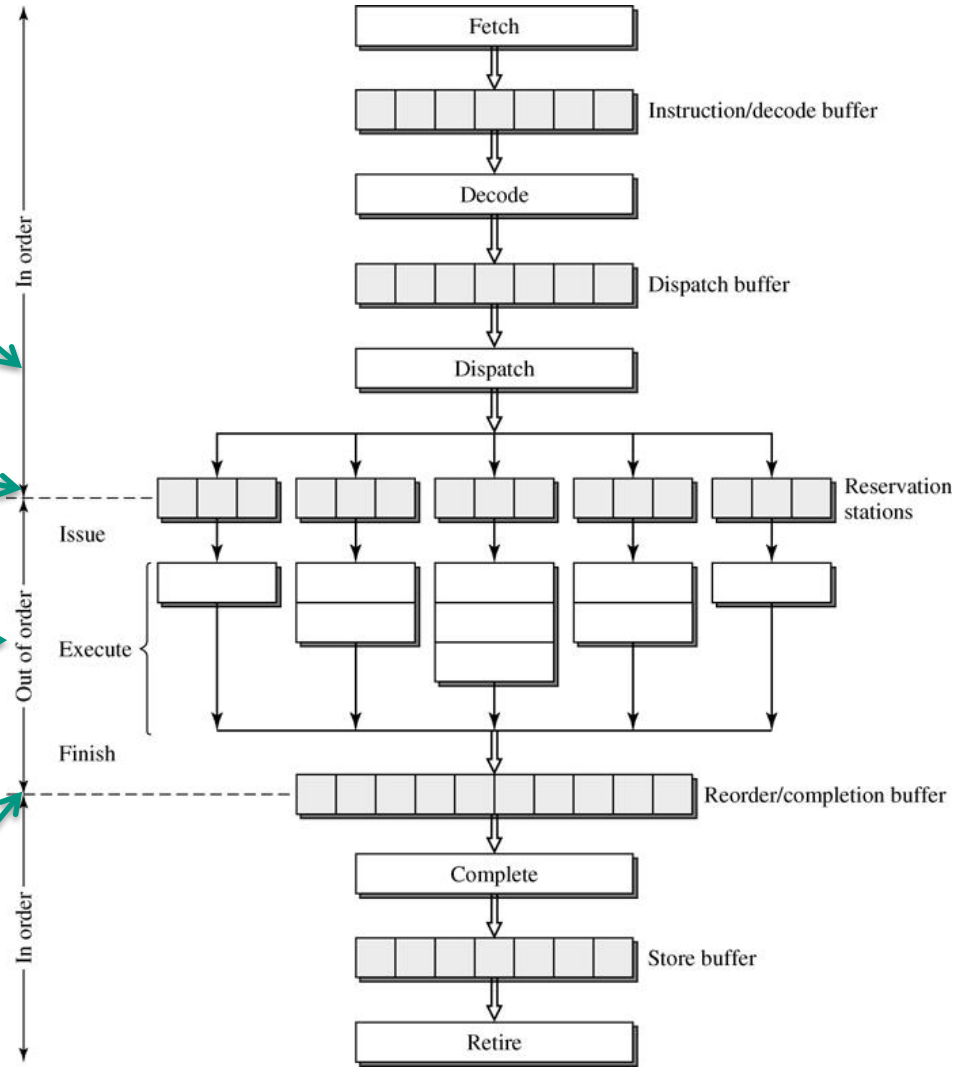
2.3.2.2 Superskalarität – Begiffe (VI)

- Zwei Befehle können parallel ausgeführt werden, wenn:
  - Beide einfache Instruktionen sind
    - z.B. keine speziellen Einheiten einer bestimmten Pipeline benötigt werden
  - Instruktion 1 kein Sprung ist
  - Wenn das Ziel von Instruktion 1 nicht die Quelle von Instruktion 2 ist
  - Wenn das Ziel von Instruktion 1 nicht das Ziel von Instruktion 2 ist

- Wenn zwei Befehle nicht parallel ausgeführt werden können:
  - Instruktion 1 in erster Pipeline abarbeiten
  - Instruktion 2 im nächsten Zyklus abarbeiten
    - Wenn möglich, zusammen mit der darauf folgenden Instruktion

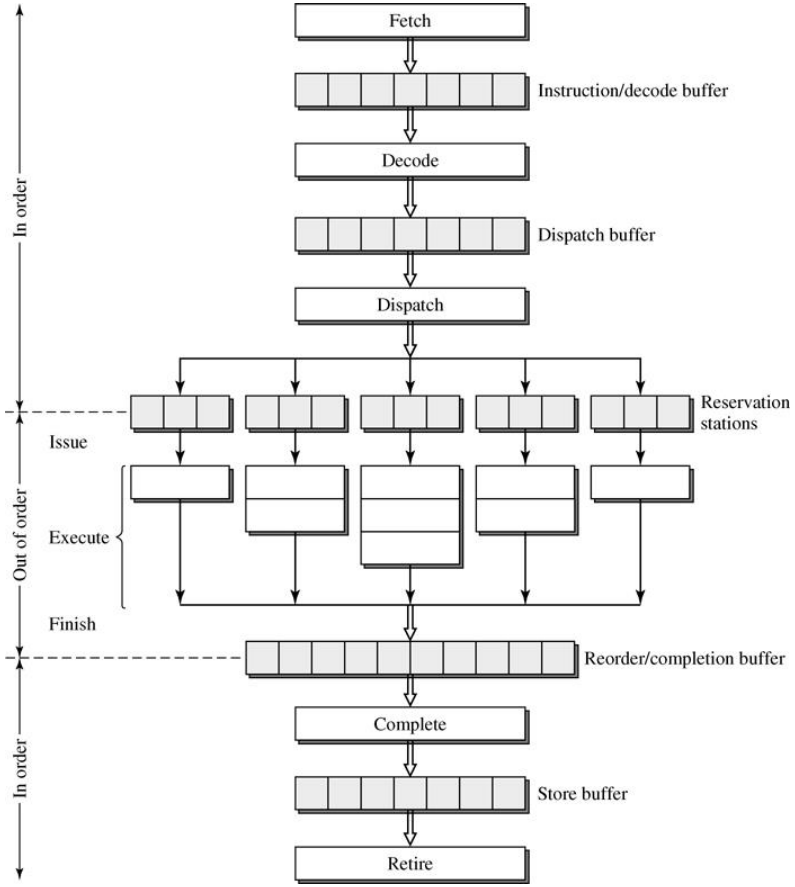
2.3.2.2 Dynamisches Scheduling - Prozessorpipeline (I)

- Prinzipielle Vorgehensweise:
  - Zuweisen von Befehlen an Ausführungseinheiten
    - In-order Issue
    - Out-of-Order Issue
  - Reservation Stations
  - Weiterleiten der Befehle an Funktionseinheiten, wenn Operanden verfügbar sind
  - Ablegen der Befehle in Programmreihenfolge in dem Reorder-Puffer



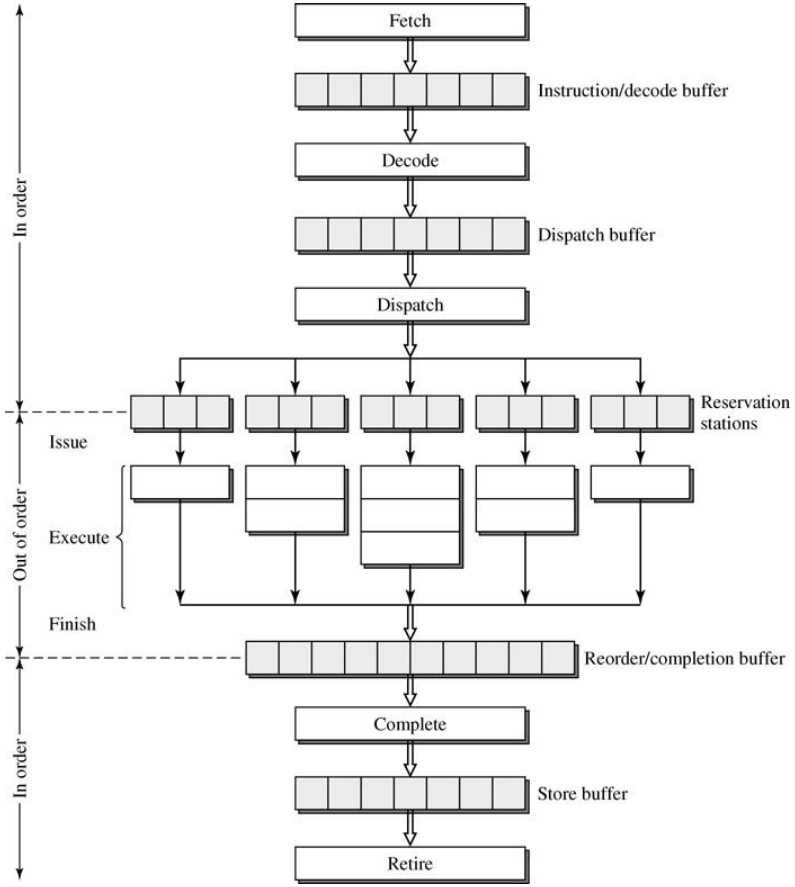
2.3.2.2 Dynamisches Scheduling - Prozessorpipeline (II)

- IF Phase
  - Holen mehrerer Befehle in den Hole-Puffer
  - Verwaltung der Komponenten für die Sprungzielvorhersage
- ID Phase
  - Dekodierung der Befehle
  - Umbenennung von Registern
  - Abbildung von logischen Registern auf physikalische Prozessorregister
  - Schreiben der Befehle ins Befehlsregister
  - Erkennen und Auflösen von Konflikten aufgrund von Daten- und Strukturabhängigkeiten



2.3.2.2 Dynamisches Scheduling – Reservation Stations

- Puffer für eine Instruktion mit ihrem Operanden
  - (siehe Tomasulo-Algorithmus, Hennessy/Patterson Buch)
- Puffer mit ein oder mehreren Einträgen. Jeder Eintrag kann eine Instruktion mit ihrem Operanden enthalten
- Konfliktauflösung mit Hilfe von Reservation Stations

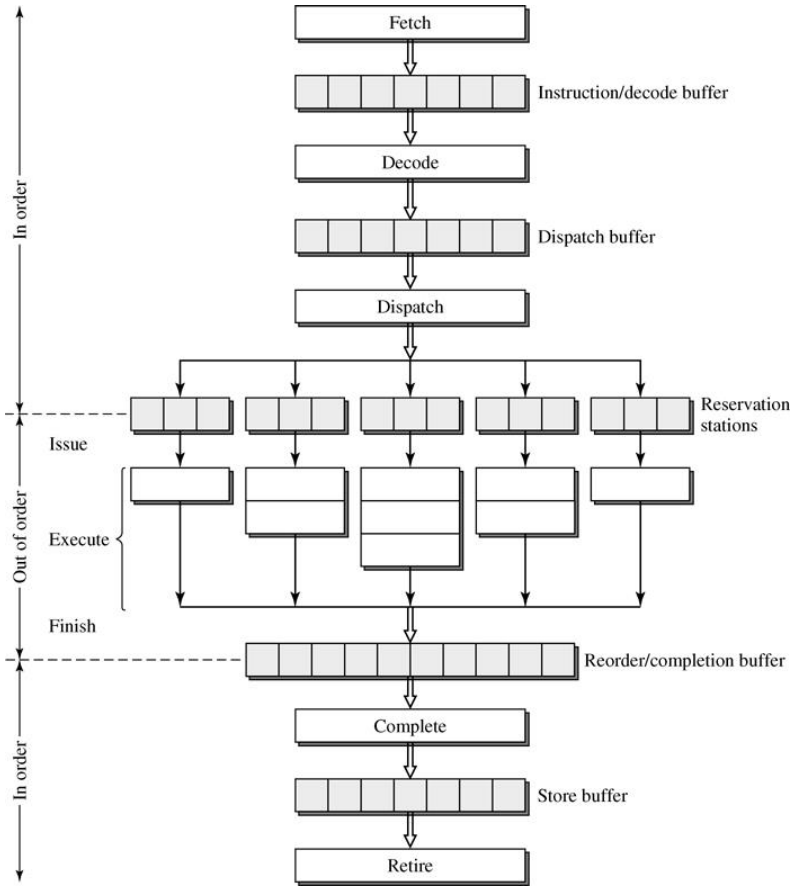


Oliver Sander
Hardware/Software Co-Design

2.3.2.2 Dynamisches Scheduling – Dispatcher/Reservation Stations

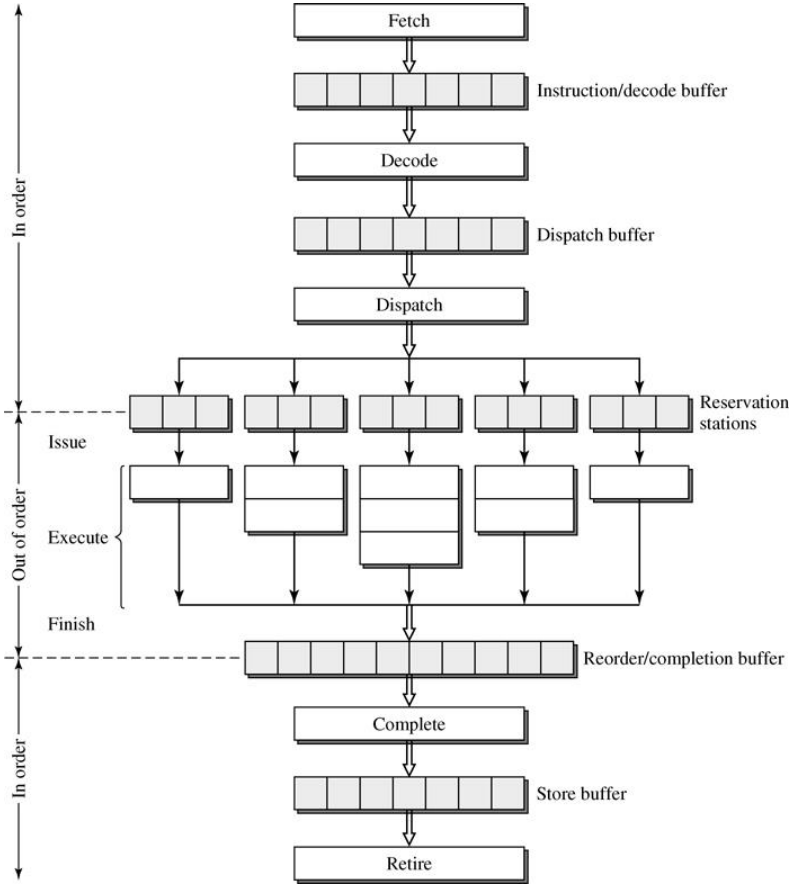
- Eine Instruktion wird aus der Reservation Station an die Funktionseinheit weitergegeben, wenn alle Operanden verfügbar sind
- Der Befehl wird in der Funktionseinheit ausgeführt
- Wenn alle Operanden verfügbar sind und die Funktionseinheit nicht besetzt ist, wird die Instruktion sofort zur Ausführung angestoßen

  → Eine Instruktion kann sich null oder mehrere Zyklen in der Reservation Station aufhalten
- Dispatch und Ausführen erfolgt nicht in der sequentiellen Programmordnung

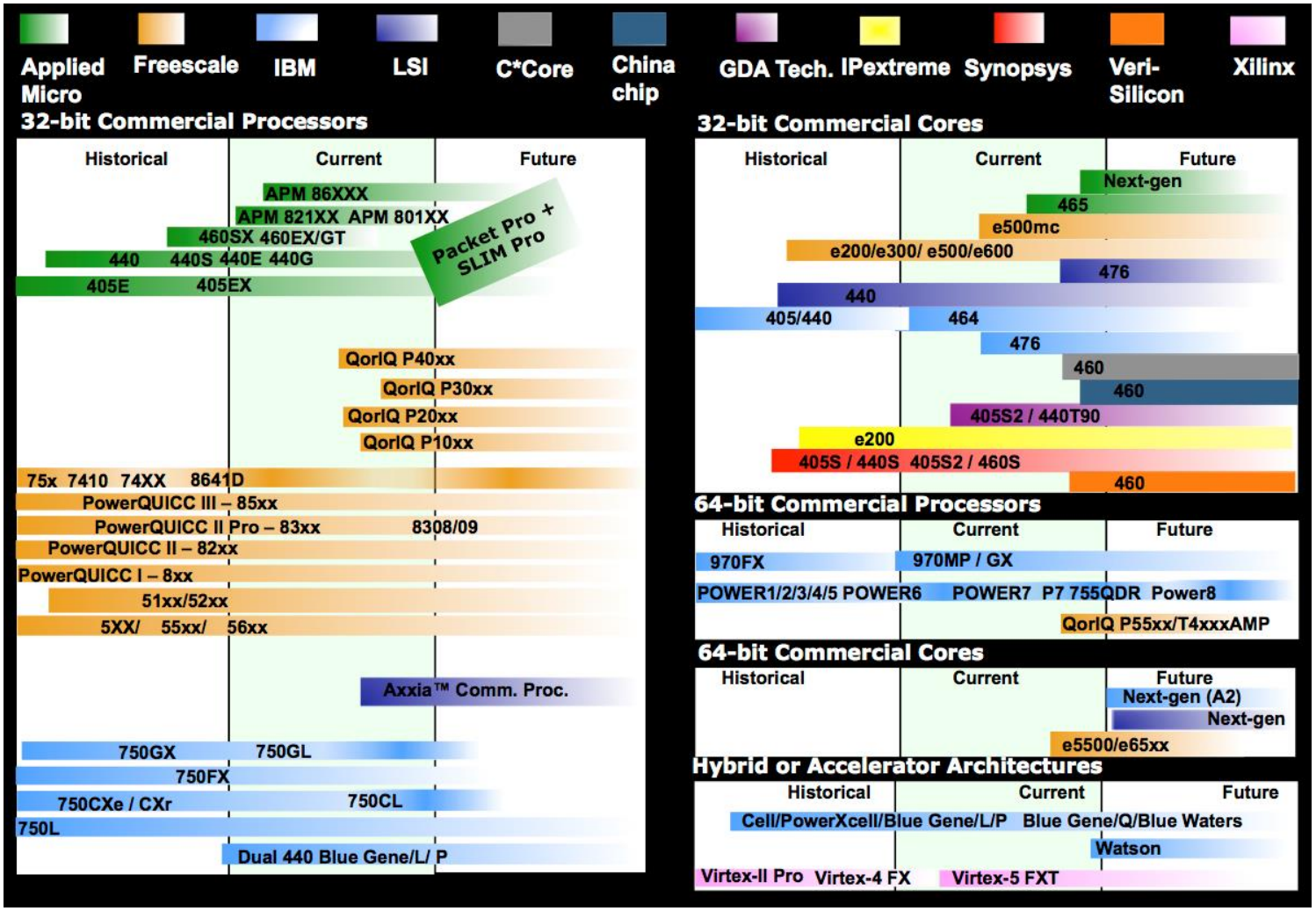


2.3.2.2 Dynamisches Scheduling - Completion-Unit

- Eine Instruktion beendet ihre Ausführung, wenn das Ergebnis für nachfolgende Befehle bereit steht (Forwarding-Puffer)
- Completion: Befehl ist vollständig
- Erfolgt unabhängig von der Programmordnung
- Aktualisierung des Zustands des Rückordnungspuffers (Reorder-Puffer)
  - Es kann eine Unterbrechung angezeigt sein
  - Es kann ein unvollständiger Befehl angezeigt werden, der von einer Spekulation abhängt



2.3.2.2 Beispiel – Power Architektur Roadmap

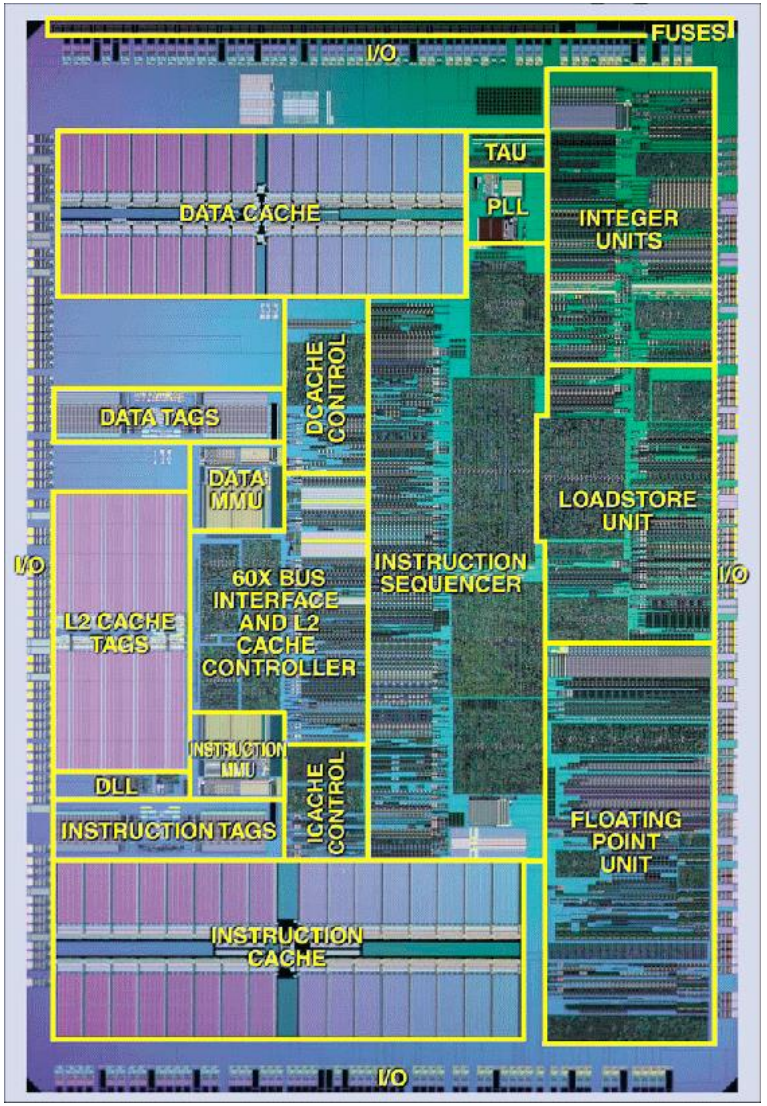


2.3.2.2 Beispiel – Power7

- 45 nm SOI process, 567 mm2
- 1.2 billion transistors
- 3.0 – 4.25 GHz clock speed
- 32+32 kB L1 instruction and data cache (per core)[13]
- 256 kB L2 Cache (per core)
- 4 MB L3 cache per core with maximum up to 32MB supported.
- max 4 chips per quad-chip module
  - 4, 6 or 8 cores per chip
    - 4 SMT threads per core
    - 12 execution units per core:
      - 2 fixed-point units
      - 2 load/store units
      - 4 double-precision floating-point units
      - 1 vector unit supporting VSX
      - 1 decimal floating-point unit
      - 1 branch unit
      - 1 condition register unit

2.3.2.2 Beispiel – PowerPC 750

- Hersteller IBM
- CMOS 0.20 µm
- Kupfer Technologie PID-8p
- 6 Metall- Lagen
- Die Size: 5.14mm x 7.78mm (40mm2)
- 6.35 Millionen Transistoren
- Logic Design: vollst. statisch
- Frequenz: 300-500 MHz
- Core- Spannung: 2 V
- I/O Spannung: 3.3V 2.5V, oder 1.8V
- Package: 25x25mm, 360 Blei-Keramik Ball Grid Array
- Leistung: 4.7 W bei 400 MHz



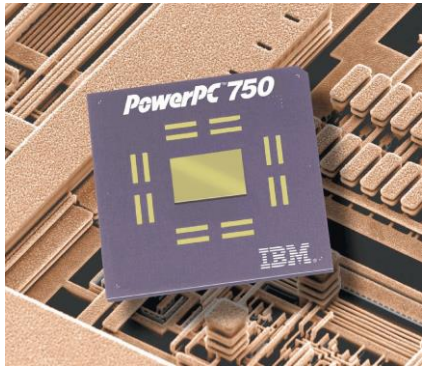
2.3.2.2 Beispiel – PowerPC 750

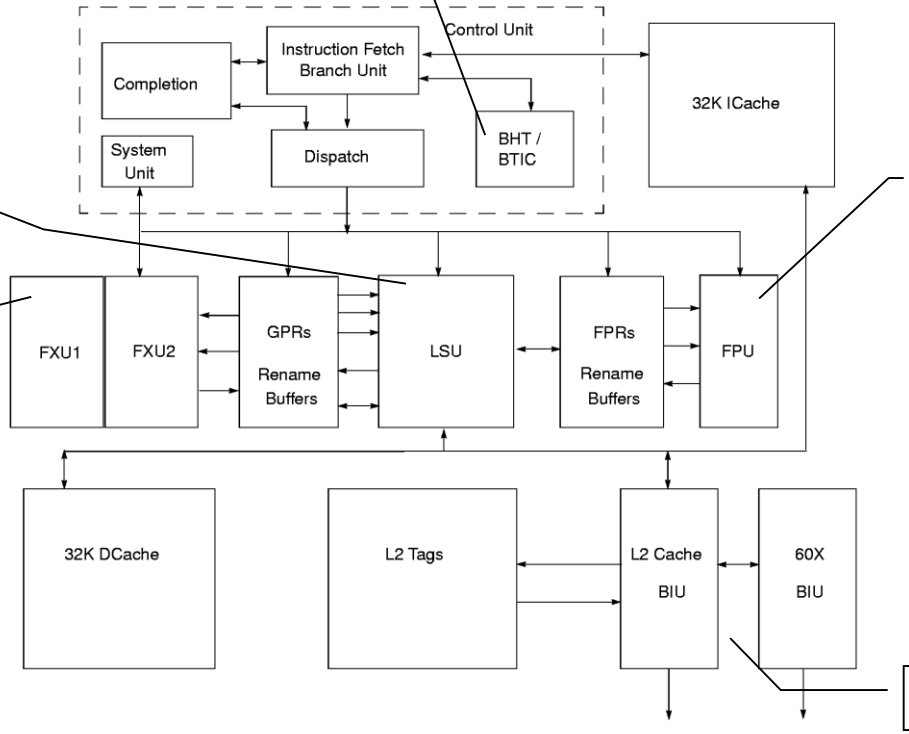
- Harvard- Architektur : getrennten Daten- / Instruktionsspeicher
- Superskalar & Out-of-order-Execution
- Spekulative Ausführung einer Instruktionssequenz

512-Entry Branch History Table for Dynamic Prediction.
Branch Target Instruction Cache

Load Store Unit

Fixed-Point Units

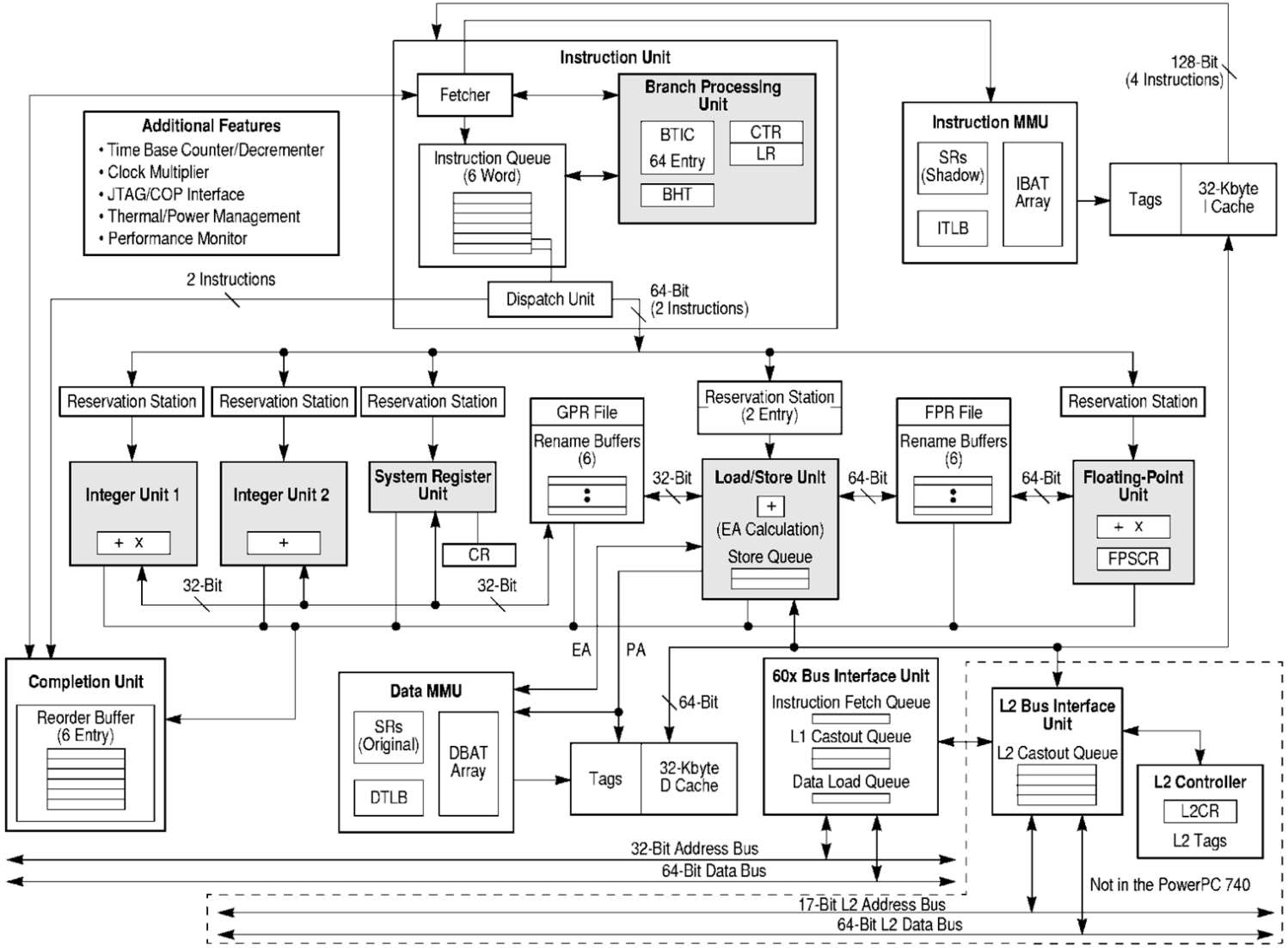




Floating-Point Unit
32 / 64 Bit

Bus Interface Unit

2.3.2.2 Beispiel – PowerPC 750



Oliver Sander
Hardware/Software Co-Design

2.3.2.2 Beispiel – PowerPC 750

- Power Management Unit
  - Statische Verlustleistung → Sleep-Modus
  - Dynamische Verlustleistung → Abschalten interner Einheiten (clock gating)
  - Integrierte Temperaturregelung Sensor + Kontrolle

- Level 2 (L2) Cache Interface
  - Interner L2-Cache Controller und 4-K entry tags
  - Unterstützt 256-Kbyte, 512-Kbyte, sowie 1-Mbyte assoziativer L2-Cache
  - Anbindung an bis zu 233 MHz externe SRAMs
  - Parity-Check im Bus-Interface

2.3.2.2 Beispiel – PowerPC 750

- Instruction Fetching & Branch Unit
  - 4 Instruktionen werden pro Taktzyklus geladen
  - 64-entry BTIC (Branch Target Instruction Cache)
  - 512-entry BHT (Branch History Table)
- Load/Store Unit (LSU)
  - alle load/store Instruktionen zwischen GPRs, FPRs und Cache/Hauptspeicher
  - effektive (logische) Adressberechnung
- Dispatch Unit
  - Hardwaredetektion von Instruktionstyp/Abhängigkeiten
  - weist 2 Instruktionen/Zyklus an 6 unabhängige Einheiten
  - 4-stufige Pipeline: fetch, dispatch, execute und complete

2.3.2.2 Beispiel – PowerPC 750

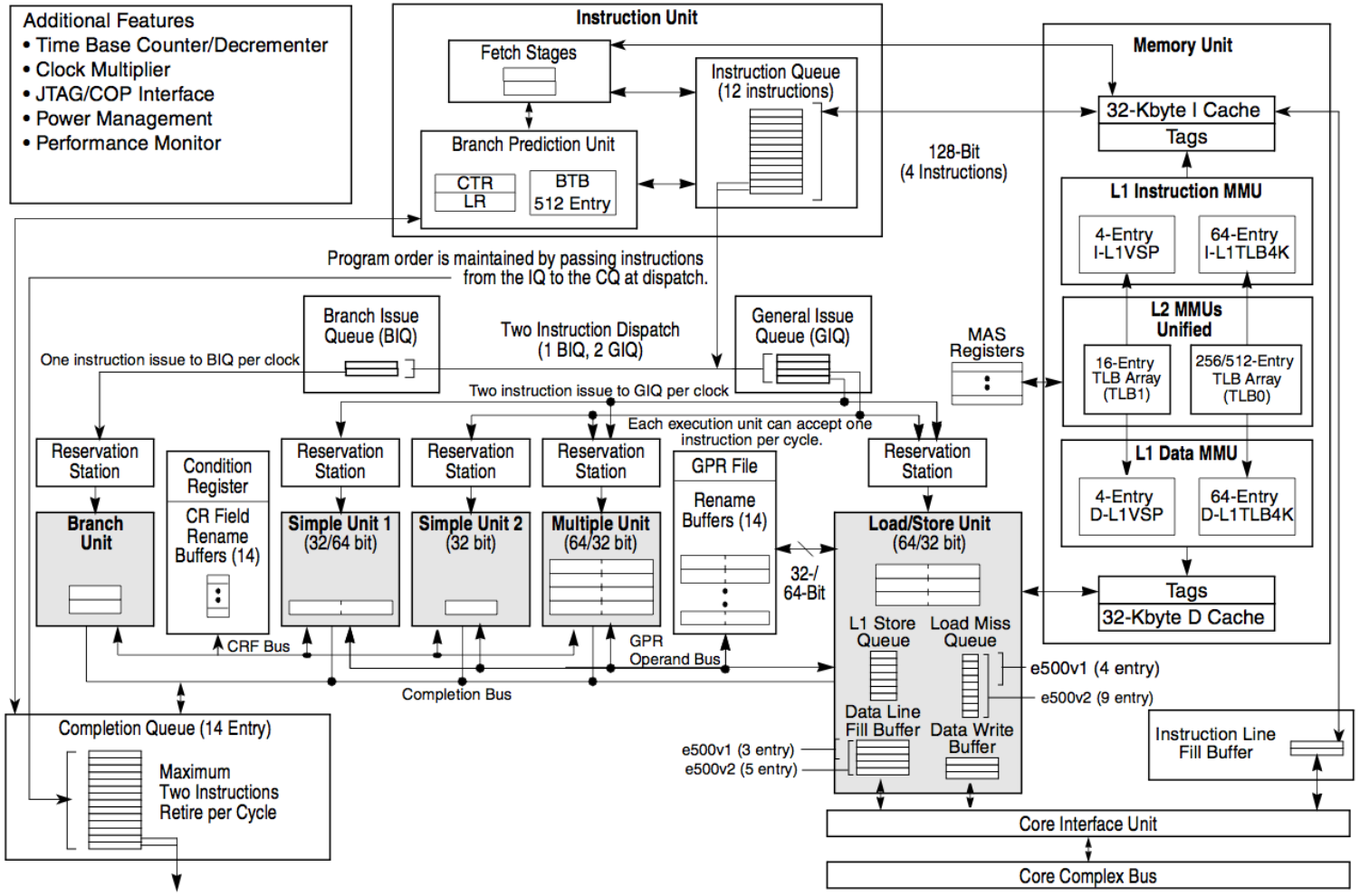
- Fixed-Point Execution Unit (FXU)
  - 1 Taktzyklus für Addition, Subtraktion, Shift, oder Rotate
  - Hardware-Multiplizierer und –Dividierer in FXU1
- Floating-Point Execution Unit (FPU)
  - optimiert für single-precision Multiplikation/Addition
  - IEEE-754 standard single- und double-precision floating-point Arithmetik
- Memory Management Unit (MMU)
  - unterstützt 52-bit virtuelle und 32-bit physikalische Adressierung
  - Hardware Adress-Translation, 128-Entry Daten- und Instruktions-TLB (Translation Lookaside Buffer)

2.3.2.2 Beispiel – PowerPC 750

- Cache Units (ICache, DCache)
  - separate 32-Kbyte 8-Wege assoziative Instruktions- und Daten-Caches
  - 3-Zustands Kohärenz (Modified, Exclusive, Invalid)
  - write-back oder write-through Daten-Cache

- Bus Interface Unit (BIU)
  - allgemeines Interface für verschiedene Systemkonfigurationen
  - 32-bit Adress - und 64-bit Daten-Bus
  - JTAG-Interface Board-Test
  - Parity-Check im Bus-Interface

2.3.2.2 Beispiel - PowerPC e500 core



2.3.2.2 Superskalarprinzip und dynamisches Scheduling - Zusammenfassung

- Aus einem sequentiellen Befehlsstrom werden Befehle zur Ausführung angestoßen (zugewiesen)
- Die Zuweisung erfolgt dynamisch durch die Hardware
- Es können mehr als ein Befehl zugewiesen werden
- Die Anzahl der zugewiesenen Befehle pro Takt wird dynamisch von der Hardware bestimmt und liegt zwischen null und der maximalen Zuweisungsbreite
- Komplexe Hardware für Zuweisungslogik erforderlich
- Mehrere von einander unabhängige Funktionsanweisungen sind verfügbar.
- Mikroarchitektur bestimmt superskalare Eigenschaft

- Was ist der Unterschied zwischen Skalar und Superskalar?
- Welche zusätzlichen Einheiten werden benötigt? Welche Funktion habe diese?
- Welche Probleme ergeben sich durch parallele Pipelines?
- Was ist dynamisches Scheduling?
- Was ist spekulative Ausführung?
- Einordnung nach Flynn?

Inhalt

2.3.2.3 Very Long Instruction Word (VLIW) (I)

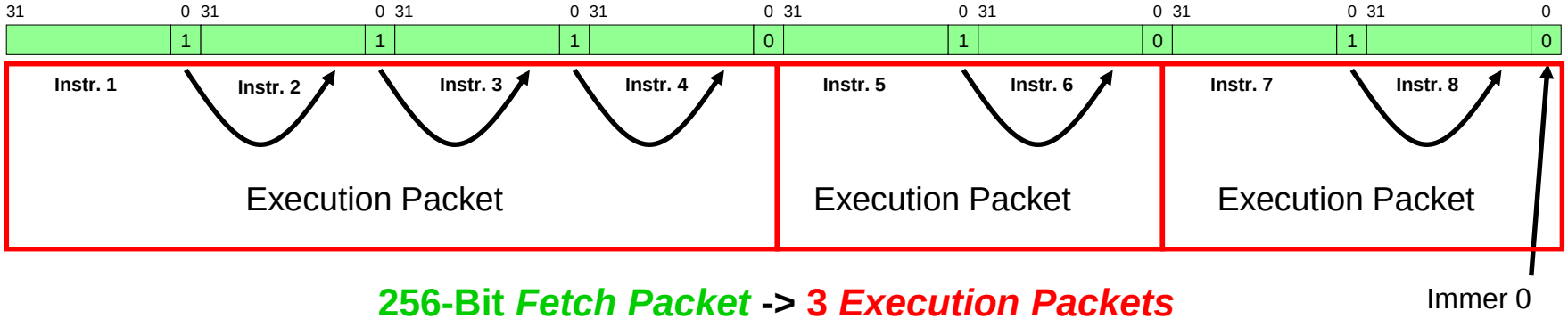
- VLIW = Very Long Instruction Word
- Architekturtechnik, bei der ein Compiler eine feste Anzahl von einfachen, voneinander unabhängigen Befehlen zu einem Befehlspaket zusammenpackt und in einem Maschinenbefehlswort meist fester Länge speichert.
- Das Maschinenbefehlsformat eines VLIW-Befehlspakets kann mehrere hundert Bits lang sein, in der Praxis sind dies zwischen 128 und 1024 Bits.
- Alle Befehle innerhalb eines VLIW-Befehlspakets müssen unabhängig voneinander sein und eigene Opcodes und Operandenbezeichner enthalten.

2.3.2.3 Very Long Instruction Word (VLIW) (II)

- Die Anzahl der Befehle in einem VLIW-Befehlspaket ist in der Regel fest.
  - Wenn die volle Bandbreite eines VLIW-Befehlspakets nicht ausgenutzt werden kann, muss es mit Leerbefehlen aufgefüllt werden.

- Problem bei VLIW-Architekturen:
  - volle Parallelität nicht immer ausnutzbar → Befehlswort teilweise unnötig lang
  - Neuere VLIW-Architekturen sind in der Lage, durch ein komprimiertes Befehlsformat auf das Auffüllen mit Leerbefehlen zu verzichten.

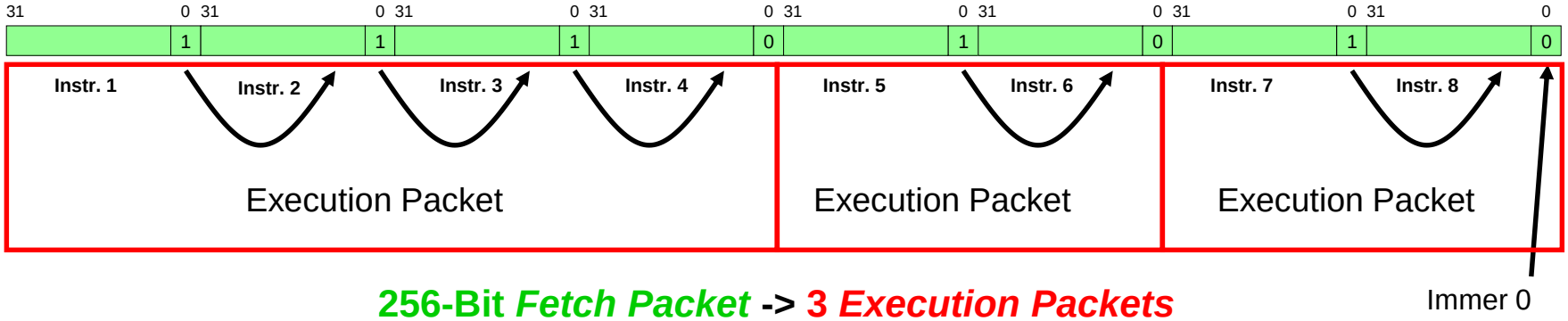
- Lösung: Instruction-Packing: Instruktionswort enthält mehrere Teil-Instruktionen



256-Bit *Fetch Packet* -> 3 *Execution Packets*

2.3.2.3 Very Long Instruction Word (VLIW) (III)

- Befehlswort enthält bis zu 8 32-Bit-Befehle (je 5 ns-Zyklus) = Befehlspaket
  - ein Bit gibt an, ob nächster Befehl zum selben Befehlspaket gehört,
  - erlaubt Befehle unterschiedlicher Bitbreite
- Teil-Instruktionen bedienen sich nebenläufiger Funktions-Einheiten.
- Komplizierte Compiler/Code-Generatoren
  - Scheduling muss konfliktfrei für die Ausführung geplant sein.



256-Bit *Fetch Packet* -> 3 *Execution Packets*

2.3.2.3 Very Long Instruction Word (VLIW) (IV)

- Vorläufer zu Superskalarität
- Geringe Codedichte
  - Compiler schwer zu entwickeln
- Vom Compiler generiertes Scheduling zur Laufzeit von Hardware durchzuführen
- VLIW vor ca. 15 Jahren mit hoher Erwartung entwickelt
  - Erreichbarer Parallelitäts-Grad für viele Anwendungen nicht konstant
  - nur für Anwendungen mit viel Parallelität sinnvoll
- Anwendungen müssen von der Struktur her geeignet sein
  - Effizienz bei hoher synchroner Parallelität auf Instruktionsebene

2.3.2.3 Very Long Instruction Word (VLIW) (V)

- Ein VLIW-Prozessor besteht aus einer Anzahl von Ausführungseinheiten, die jeweils eine Maschinenoperation taktsynchron zu den anderen Maschinenoperationen eines VLIW-Befehlspakets ausführen können, wobei ein VLIW-Befehlspaket maximal so viele einfache Befehle umfasst, wie Ausführungseinheiten in dem VLIW-Prozessor vorhanden sind.
- Der Prozessor startet im Idealfall in jedem Takt ein VLIW-Befehlspaket.
- Die Befehle in einem solchen VLIW-Befehlswort werden dann gleichzeitig geholt, decodiert, zugewiesen und ausgeführt.

2.3.2.3 Very Long Instruction Word (VLIW) (VI)

- In Abhängigkeit von der Anzahl n der Befehle, die gemeinsam durch die Pipeline fließen und entsprechend der maximalen Anzahl n von Befehlen in einem Befehlspaket spricht man von einem n-fachen VLIW-Prozessor.

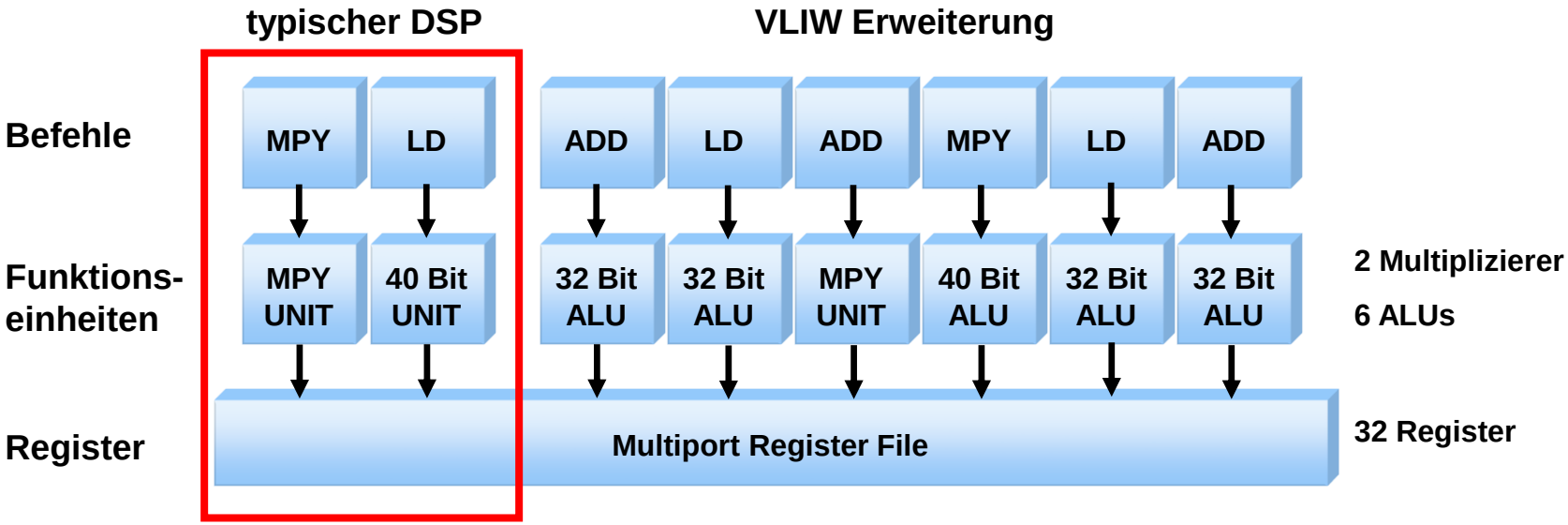
- Ablaufreihenfolge der VLIW-Befehlspakete ist fest, Ausführung auch außerhalb der Programmreihenfolge nicht möglich.

- Operanden stehen in einem allen Ausführungseinheiten zugänglichen Registersatz.

2.3.2.3 Beispiel TI TMS 320C62x

- TI DSP TMS 320C62x-Familie mit VLIW-Architektur
  - 1997/98: schnellste DSP-Familie mit ca. 1600 MIPS, 200 MHz / 5 ns Zykluszeit

- Verlagerung der Komplexität: von HW in optimierende Compilerschritte
  - zur Übersetzungszeit parallel ausführbare Rechenoperationen bestimmen
  - einfachere Prozessoren (Fläche):
    - TMS320C6201: 550.000 Transistoren « Pentium: 5.000.000 Transistoren



2.3.2.3 VLIW-Befehl vs. CISC und SIMD-Befehl

- Unterschied zu einem CISC-Befehl:
  - CISC-Befehl kann mit einem Opcode mehrere, eventuell sequenziell nacheinander ablaufende Operationen codieren

- Unterschied zu einem SIMD-Befehl:
  - SIMD-Befehle sind die Multimediabefehlen, bei denen ein Opcode eine gleichartige Operation auf einer Anzahl von Operanden(paaren) auslöst.
  - Die Operationen innerhalb eines VLIW-Befehlspakets sind in der Regel verschiedenartig.

- Was ist der Unterschied zu Skalar und Superskalar?
- Wo findet die Parallelisierung statt?
- Wie unterscheidet sich die Instruktion?
- Wie muss die Hardware aufgebaut sein?
- Einordnung nach Flynn?

Inhalt

2.3.2.4 Single Instruction Multiple Data (SIMD)

- Gleiche Instruktion auf mehreren Daten
  - Multimediaanwendungen
    - Viele Daten gleichzeitig
    - Viele Parallelisierungsmöglichkeiten
- Reduziert Zyklenzahl für gleiche Menge von Daten
- Vektorisierung von Daten
  - Parallele Berechnung der Vektoren
  - Speichern der einzelnen Daten in Register
- Schwer zu implementieren
  - Schwer automatisch Code zu finden, der SIMD gut ausnutzt
    - Code von Hand schreiben/optimieren

2.3.2.4 SIMD – Multimedia-Erweiterungen

- Multimedia-Anwendungen
  - z.B.: Audio/Video Playback, DVD, Videokonferenzen
  - 8/16-Bit Datentypen (z.B.: Pixeldarstellung in Bildverarbeitung)
  - viele arithmetische Operationen (Multiplikation, Addition)
  - große Datenmengen, große I/O Bandbreite
  - viel Datenparallelität → Single Instruction Multiple Data (SIMD)

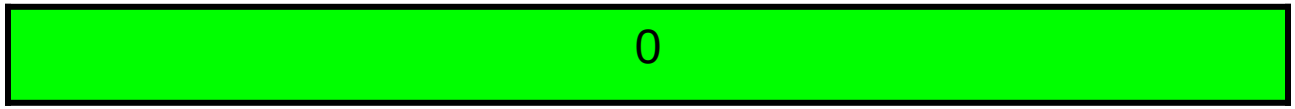
- Sub-Word Execution Model
  - 32/64-Bit Register und ALUs in kleinere Einheiten auftrennen
  - Instruktionen arbeiten parallel auf den Einheiten -> SIMD

- Beispiele:
  - MMX (Intel), SSE (Intel), VIS (UltraSparc), MDMX (MIPS), MAX-2 (HP)

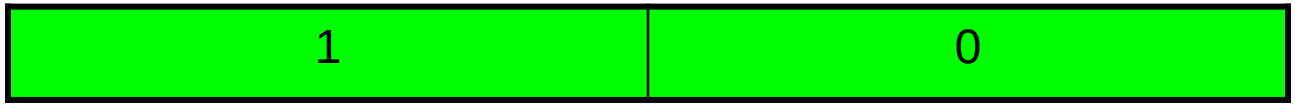
2.3.2.4 SIMD – Sub-Word Execution Model (I)

- Beispiel: 64-Bit Datentyp in kleinere Einheiten auftrennen
- Instruktionen parallel auf kleineren Ausführungseinheiten

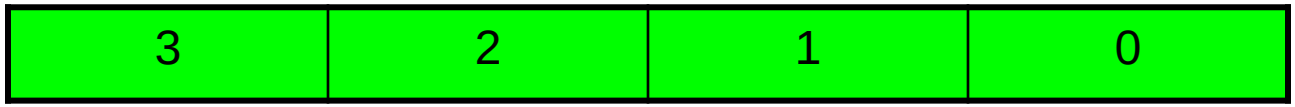
- Long Word



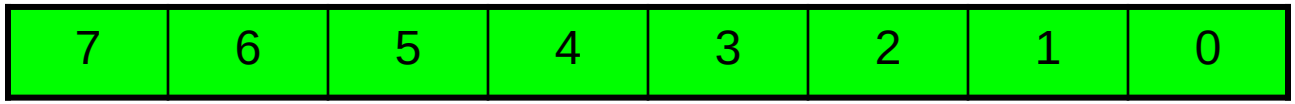
- Packed Word



- Packed Half-Word



- Packed Byte

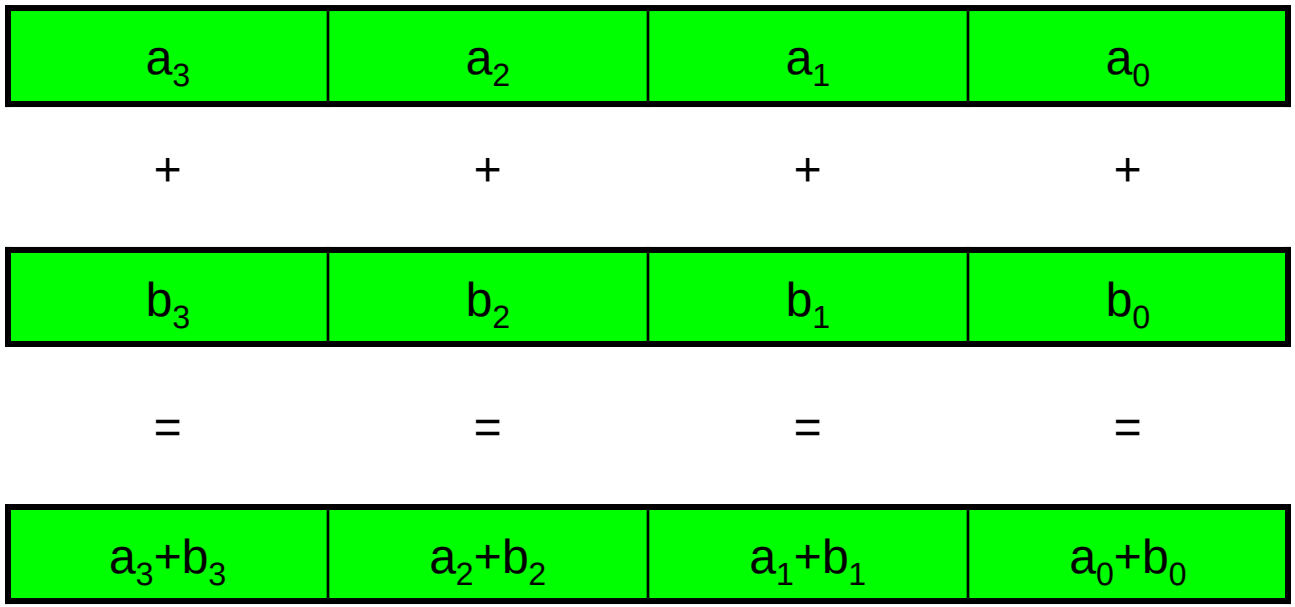


2.3.2.4 SIMD – Sub-Word Execution Model (II)

- Beispiel 1: Addition
  - Instruktionen parallel auf 4 Ausführungseinheiten

- ADD R3 := R1, R2

- R1



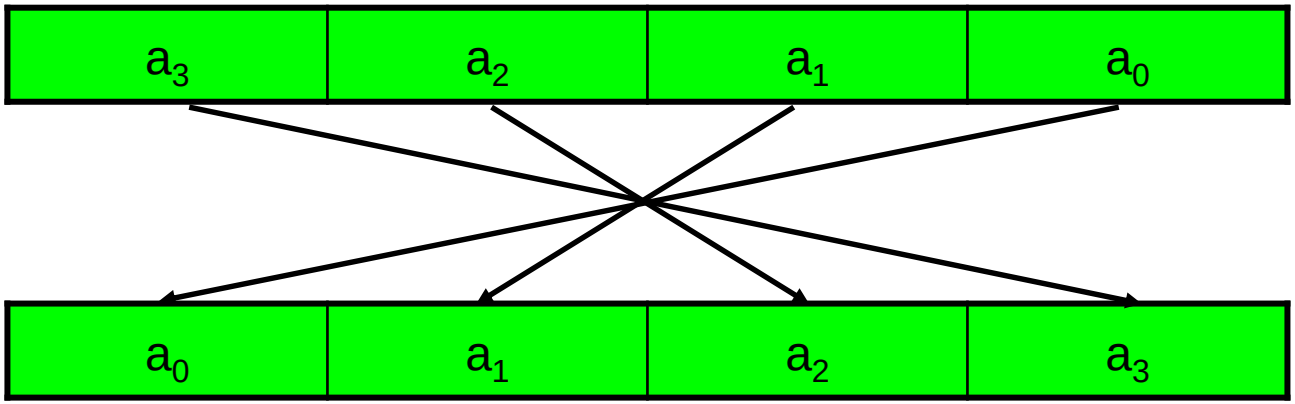
- R2
- R3

2.3.2.4 SIMD – Sub-Word Execution Model (III)

- Beispiel 2: Permutation
  - Vertauschen der Reihenfolge der Half-Words

- PERMUTE R3 := R1 (Muster 0123)

- R1
- R3



- Welche Funktion haben SIMD-instruktionen?
- Welche Parallelität wird ausgenutzt?
- In welchen Anwendungen macht SIMD Sinn?
- Welche bekannten Beispiele gibt es für SIMD-Funktions-Einheiten?