UR

Informationstechnische Grundlagen

Rechnerarchitektur und Assembler

Informationswissenschaft
Universität Regensburg
Jürgen Reischer

Grundlagen

- Überblick:
  - Rechnerarchitektur:
    - Grundaufbau von Rechnern (Zentraleinheit und Peripherie);
    - Verarbeitungseinheiten und -mechanismen eines Prozessors;
    - Maßzahlen für Rechnerleistung und Herstellungsprozesse.
  - Assembler:
    - Aufbau des Maschinenkodes (Instruktionssatzes) eines Prozessors;
    - einfache Programmierung in Maschinenkode.

Grundlagen

- Alle heutigen Computer arbeiten nach dem grundlegenden EVA-Prinzip:
  - *E* = *Eingabe*: Daten werden über eine Eingabeeinheit eingelesen (Eingabeschnittstelle);
  - *V* = *Verarbeitung*: Daten werden über eine Verarbeitungseinheit verrechnet und/oder in einer Speichereinheit zwischengespeichert (um später wiederverwendet werden zu können);
  - *A* = *Ausgabe*: Daten werden über eine Ausgabeeinheit weiter- oder zurückgegeben (Ausgabeschnittstelle).

Grundlagen

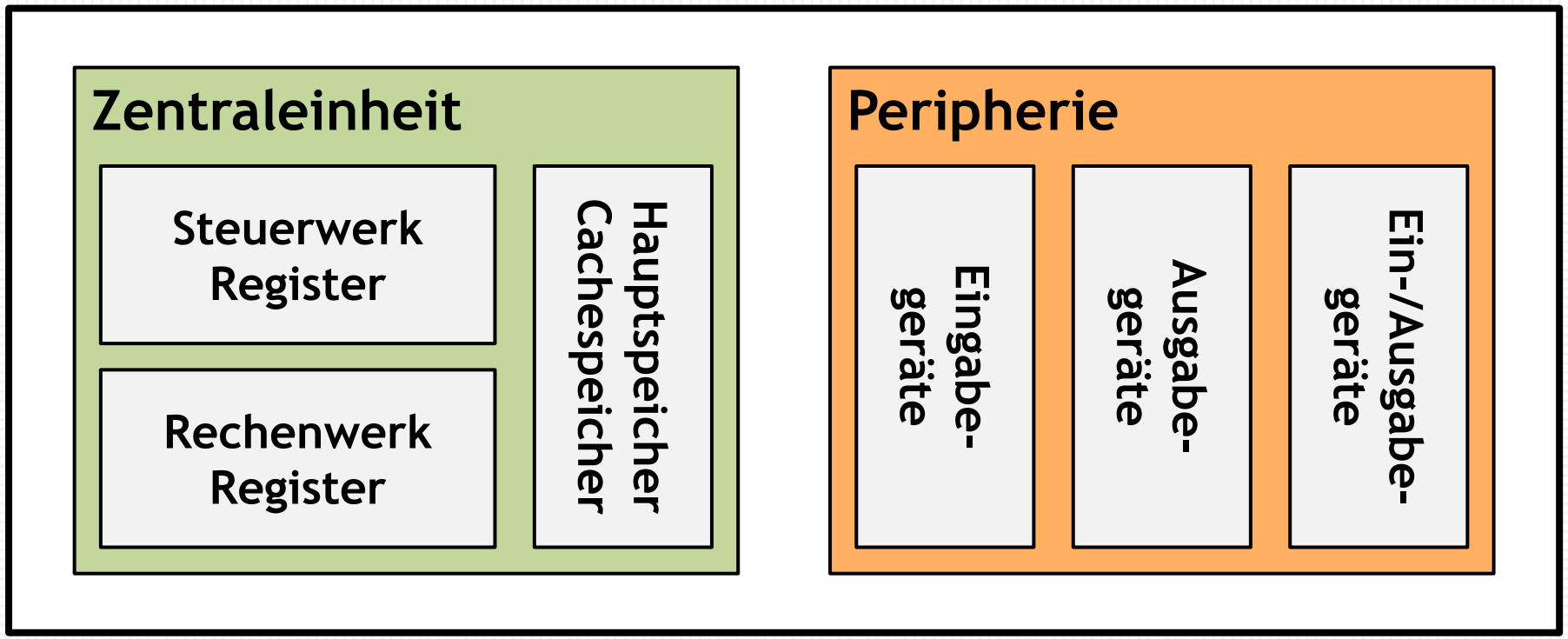
- Das EVA-Prinzip findet sich auf allen Hard- und Software-Ebenen wieder:
  - Hardware:
    - elementares Schaltelement, das eine arithmetische oder logische Funktion repräsentiert (¬X, X∨Y, X+Y usw.);
    - Sensor-Daten (Reiz, Input) ⇨ Aktor-Daten (Reaktion, Output).
  - Software:
    - Eingabe durch Nutzer, Verarbeitung/Ausgabe durch Rechner;
    - Funktion in einer Programmiersprache: Y := F(X), Z := G(A,B)
      - Argumente/Parameter (Eingabe): X bzw. A, B;
      - Abbildung von X nach Y bzw. A, B nach Z (Verarbeitung): F, G;
      - Resultat/Funktionsergebnis (Ausgabe): Y bzw. Z.

Rechnerarchitektur

- Computer folgen hierfür einem bestimmten Aufbauprinzip auf der Ebene der Hardware (im Gegensatz zur Software):
  - *Von-Neumann-Rechner*: Daten und Programme (Instruktionen, Programmkode) werden gemeinsam in einem einzigen unspezifischen Speicher verwaltet und verarbeitet;
  - *Harvard-Architektur*: Daten und Programme werden getrennt verwaltet und verarbeitet (heute nurmehr in bestimmten Bereichen zu finden).

Rechnerarchitektur

- Im Detail besteht ein Rechner nach dem Von-Neumann-Prinzip aus folgenden Komponenten:

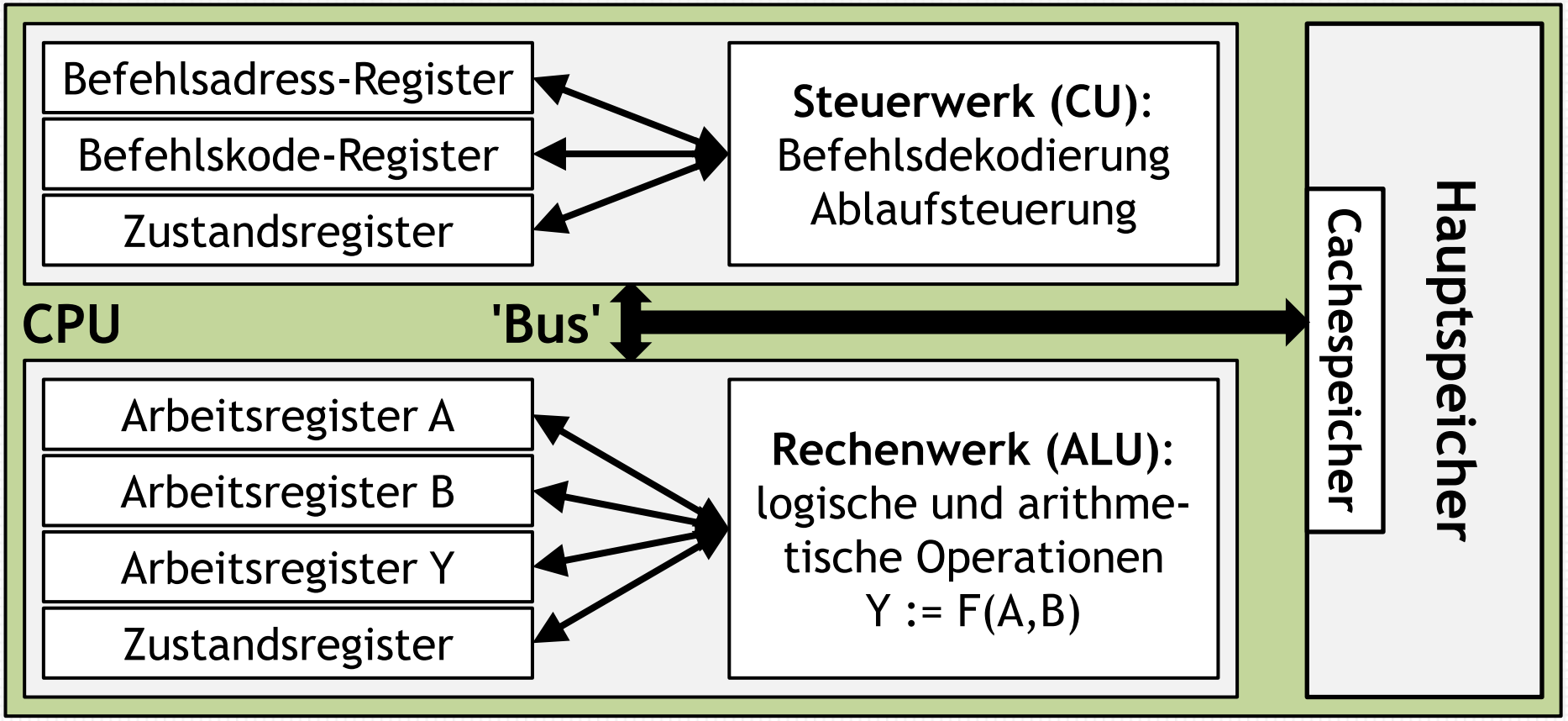


Rechnerarchitektur – Zentraleinheit

- Zentraleinheit:
  - Prozessor (CPU = 'Central Processing Unit'): Zentrale Recheneinheit, in der alle Steuer- und Rechenoperationen durchgeführt werden:
    - Steuerwerk (CU = 'Control Unit'): Durchführung der Ablaufkontrolle eines Programms (s. Details unten);
    - Rechenwerk (ALU = 'Arithmetic-Logic Unit'): Durchführung arithmetischer und logischer Operationen (s. Details unten).
  - Speicher:
    - Hauptspeicher: großer, aber langsamer Daten- und Instruktions-speicher;
    - Nebenspeicher: schnelle(r), aber kleine(r) 'Cache'-Speicher, die Zugriffe auf den langsamen Hauptspeicher puffern.

Rechnerarchitektur – Zentraleinheit

Zentraleinheit mit CPU und Speicher im Überblick:



Register: 'Zwischenablage' für Berechnungen und Ergebnisse.

Rechnerarchitektur – Zentraleinheit

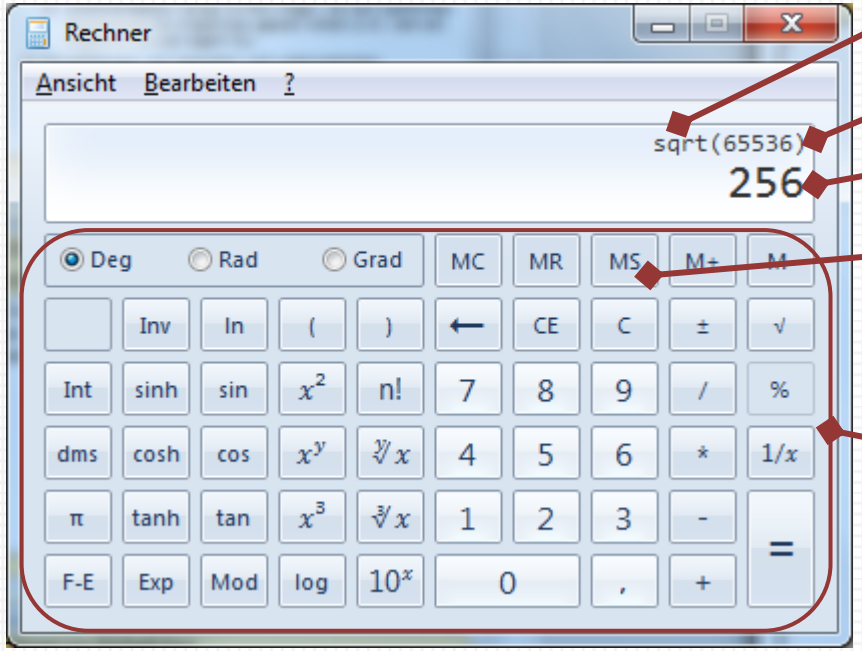
- Steuerwerk:
  - Satz von Registern für Zwischenspeicherung von Werten:
    - Befehlsadress-Register: Speicheradresse des nächsten auszuführenden Befehls;
    - Befehlskode-Register: aktuell dekodierter und ausgeführter Befehl;
    - Zustandsregister: Informationen über den Verarbeitungszustand des aktuellen Befehls (z. B. Phase der Dekodierung).
  - Regelung des Verarbeitungsflusses (Ablaufkontrolle):
    - Holen und Dekodieren des nächsten Befehls in das Befehlskode-Register;
    - Weiterschalten des Befehlsadress-Registers (Setzen der Befehls-adresse auf den nächsten Befehl);
    - Setzen des Zustandsregisters.

Rechnerarchitektur – Zentraleinheit

- Rechenwerk:
  - Satz von Registern für Zwischenspeicherung von Werten:
    - Arbeitsregister: prozessor-interne Speicherplätze, die als Puffer für zu verarbeitende bzw. verarbeitete Daten fungieren (meist für Input- und Output-Daten gleichermaßen verwendbar);
    - Zustandsregister: Menge von Bits ('Flags'), die nach bestimmten Operationen bzw. Ergebnissen gesetzt werden (z. B. wenn ein Ergebnis null oder negativ ist).
  - Durchführung von Berechnungen:
    - arithmetische und logische Operationen für Ganz- und Bruchzahlen (Fließkommazahlen);
    - Setzen des Statusregisters.

Rechnerarchitektur – Zentraleinheit

Die CPU kann man sich ähnlich einem Taschenrechner vorstellen, der nicht vom Menschen, sondern vom Steuerwerk bedient wird:



ALU-Operation F := √A

Input-Register A = 65536

Output-Register Y = 256

Hintergrundspeicher

Steuerkontrolle ('Programm'):
1. Eingabe 65536
2. Eingabe √
3. Return
4. Ausgabe 256

Rechnerarchitektur – Zentraleinheit

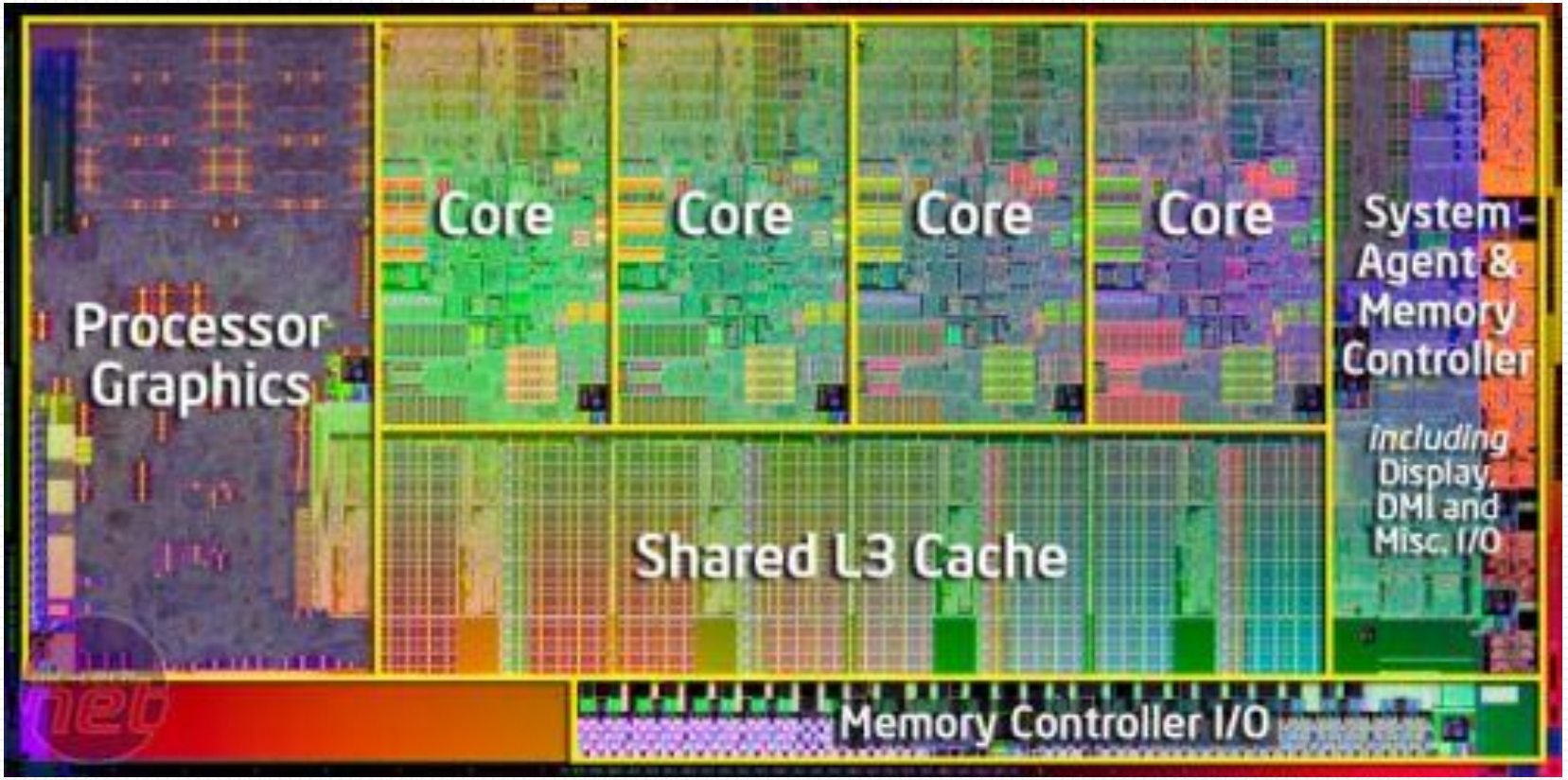
- Unterscheidungsmerkmale für CPUs:
  - CISC- vs. RISC-CPU:
    - RISC = 'Reduced Instruction Set Computer':
      - CPUs mit elementaren, einfach und schnell abzuarbeitenden Befehlen in Maschinensprache (native Sprache eines Prozessors);
      - eine bestimmte Aufgabe muss durch mehrere Befehle umgesetzt werden, die jedoch sehr schnell verarbeitet werden können.
    - CISC = 'Complex Instruction Set Computer':
      - CPUs mit elementaren und komplexen Maschinensprach-Befehlen, die mehrere elementare Befehle zusammenfassen;
      - eine bestimmte Aufgabe benötigt oft nur einen einzigen Befehl, der jedoch vergleichsweise langsam ausgeführt wird.

Rechnerarchitektur – Zentraleinheit

- Ein- vs. Mehrkern-CPUs:
  - Einkern-CPU ('Singlecore-CPU'):
    - besitzt eine einzige Zentraleinheit, die seriell alle Aufgaben ('Tasks') erledigen muss;
    - jeder Kern kann genau einen solchen Prozess (= Task) zu einer Zeit durchführen (daher der Name 'Prozessor' für die CPU).
  - Mehrkern-CPU ('Multicore-CPU'):
    - besitzt mehrere Zentraleinheiten, die parallel mehrere Aufgaben erledigen können (typisch derzeit 2, 4, 6 oder 8 Kerne; zukünftig auch 16, 32, 64 Kerne usw.);
    - jeder Kern kann grundsätzlich unabhängig von allen anderen Kernen betrieben werden, wobei Kerne zum Zwecke des Stromsparens auch vollständig abgeschaltet werden können.

Rechnerarchitektur – Zentraleinheit

Bild einer Vierkern-CPU:



http://www.ozone3d.net/public/jegx/201101/intel-sandy-bridge-i7-2600-arch.jpg (2.1.2012)

Rechnerarchitektur – Zentraleinheit

- CPU mit Single- vs. Multi-/Hyperthread-Kernen:
  - CPU mit Single-Thread-Kern: Jeder CPU-Kern kann zu einer Zeit genau eine Aufgabe erledigen:
    - analog einem einzigen Programmfaden ('Thread'), der vom Anfang bis zum Ende durchläuft (Thread = 'leichtgewichtiger' Prozess);
    - nur jeweils eine einzige Funktionseinheit des Kerns ist durch eine Aufgabe ausgelastet, alle anderen liegen brach.
  - CPU mit Multi-/Hyperthread-Kern: Jeder CPU-Kern kann zwei oder mehr Aufgaben zur gleichen Zeit auf dem selben Kern erledigen:
    - analog mehreren Programmfäden, die parallel vom jeweiligen Anfang bis zum jeweiligen Ende unabhängig durchlaufen;
    - verschiedene Funktionseinheiten ein und des selben CPU-Kerns können im Idealfall gleichzeitig ausgelastet werden.

Rechnerarchitektur – Zentraleinheit

- Nonpipelining- vs. Pipelining-CPU:
  - Nonpipelining: Einzelne Befehle werden vollständig seriell abgearbeitet, bevor der nächste Befehl ausgeführt wird:
    - analog einem Werkstück, das erst fertiggestellt wird, bevor das nächste begonnen werden kann ('eins nach dem andern');
    - Werkstattmetapher: Allround-Handwerker, der alle Tätigkeiten vom Anfang bis zum Ende vollständig selbst durchführt.
  - Pipelining: Befehle werden parallel-überlappend abgearbeitet, d. h. mehrere Befehle werden zugleich ausgeführt:
    - analog mehreren Werkstücken, die in mehr oder weniger fertigen Produktionsstadien zugleich verarbeitet werden;
    - Fließbandmetapher: mehrstufiges Fertigstellungssystem, an dem pro Stufe ein hochspezialisierter 'Handwerker' arbeitet.

Rechnerarchitektur – Zentraleinheit

Nonpipelining:

	T 1	T 2	T 3	T 4	T 5	T 6	T 7	T 8	T 9	T 10
B 1	CALL	DECO	READ	CALC	WRIT					
B 2						CALL	DECO	READ	CALC	WRIT

Pipelining:

	T 1	T 2	T 3	T 4	T 5	T 6	T 7	T 8	T 9	T 10
B 1	CALL	DECO	READ	CALC	WRIT					
B 2		CALL	DECO	READ	CALC	WRIT				
B 3			CALL	DECO	READ	CALC	WRIT			
B 4				CALL	DECO	READ	CALC	WRIT		
B 5					CALL	DECO	READ	CALC	WRIT	
B 6						CALL	DECO	READ	CALC	WRIT

Rechnerarchitektur – Zentraleinheit

- Abkürzungen: T = Takt, B = Befehl.
- Phasen der Befehlsausführung:
  - CALL: Abrufen/Bereitstellen des nächsten auszuführenden Befehls (aus dem Speicher);
  - DECO: Dekodieren des Befehls (d. h. syntaktisch-semantische Analyse des Befehls durchführen);
  - READ: Lesen des oder der Operanden aus dem Speicher oder einem Register (Daten-Input: Ermitteln der zu verarbeitenden Werte);

  } Eingabe

  - CALC: Berechnung des Ergebnisses in der ALU (Ausführung einer Operation auf den Operanden);

  } Verar-beitung

  - WRIT: Zurückschreiben des Ergebnisses der Berechnungsoperation in den Speicher oder ein Register (Daten-Output).

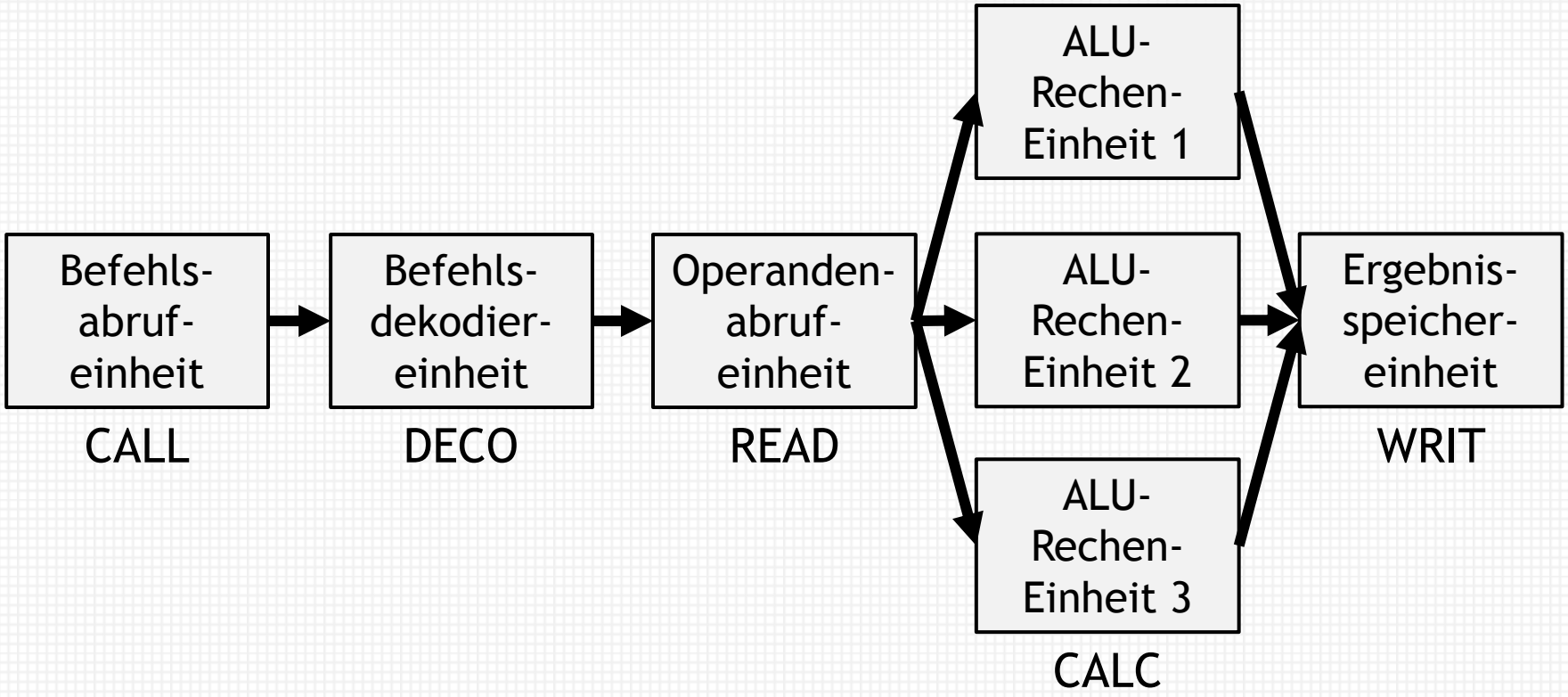
  } Ausgabe

Rechnerarchitektur – Zentraleinheit

- Superskalare vs. (sub)skalare CPU:
  - (sub)skalare CPU:
    - pro Pipeline wird nur eine Funktions- bzw. Ausführungseinheit zur Verfügung gestellt;
    - pro Takt kann somit maximal ein einzelner Befehl begonnen und fertiggestellt werden.
  - superskalare CPU:
    - pro Pipeline werden mehrere Funktions- bzw. Ausführungs-einheiten zur Verfügung gestellt;
    - pro Takt kann somit oftmals mehr als ein Befehl begonnen und fertiggestellt werden (da mehrere Ausführungseinheiten innerhalb einer Pipeline gleichzeitig tätig sind).

Rechnerarchitektur – Zentraleinheit

Schema einer superskalaren CPU mit einer Pipeline:



Rechnerarchitektur – Peripherie

- Peripheriekomponenten sind alle Bestandteile eines Rechners, die nicht zur Zentraleinheit gehören:
  - stärker mit Zentraleinheit verwobene Komponenten:
    - interne Speichermedien: (fest eingebaute) Speicherkarten, BIOS ('Basic Input-Output System' mit grundlegenden Dienstprogrammen und Grundeinstellungen [zum Booten] für einen Rechner);
    - Erweiterungen: Grafikprozessor GPU = 'Graphics Processing Unit' (heute zunehmend in CPU integriert), Soundkarte, Modem/Netzwerkkarte, Geräteanschlüsse/Schnittstellen (USB-Ports usw.).
  - weniger stark an Zentraleinheit gebundene Komponenten:
    - externe Speichermedien: Festplatte (HD/SSD), DVD-/BD-Laufwerk;
    - Ein-/Ausgabemedien: Tastatur, Touchpad, Maus, Kamera, Mikrofon, Lautsprecher, Bildschirm; Scanner, Drucker.

Rechnerarchitektur – Peripherie

- Speichermedien werden nach verschiedenen Kriterien klassifiziert:
  - *Kapazität*: Anzahl speicherbarer Daten in Bytes;
  - *Effizienz*: Geschwindigkeit des Zugriffs auf beliebige Daten;
  - *Speichermethode*: physikalisches Verfahren des Lesens und/oder Schreibens von Daten;
  - *Integriertheit*: Wechselbarkeit des eigentlichen Speichermediums;
  - *Veränderbarkeit*: Les- und Schreibbarkeit von Daten.

Rechnerarchitektur – Peripherie

- Kapazität und Effizienz stehen meist in einem umgekehrten Verhältnis zueinander ('Speicherhierarchie'):

Rechnerarchitektur – Peripherie

- Speichermethode:
  - *elektr(on)isch*: flüchtige oder nicht-flüchtige Aufladung elektronischer Bauteile wie Transistoren (z. B. Flash-Speicherchips, DRAM-Speicherchips);
  - *magnetisch*: Magnetisierung einer Scheibenoberfläche, die durch einen Schreib-Lese-Kopf ermittelt werden kann (z. B. Festplatten und Disketten);
  - *optisch*: Veränderung des Reflexionsverhaltens eines Lichtstrahls auf einer Scheibenoberfläche, was durch einen Laser abgetastet und festgestellt werden kann (z. B. DVDs/BDs).

Rechnerarchitektur – Peripherie

- Integriertheit:
  - Festmedien:
    - CPU-intern: Register, Cache (Level 1/2/3);
    - CPU-extern: (eingebauter/aufgelöteter) Hauptspeicher, BIOS;
    - HD/SSD/Flash-Speicher (sofern fest verbaut);
    - Barcode, RFID ('Radio Frequency Identification'), QR-Tags ('Quick Response Code').

http://de.wikipedia.org/wiki/QR-Code (3.1.2012)

  - Wechselmedien:
    - externe/r HD/Flash-Speicher;
    - DVD/BD (ausschließlich Wechselmedien);
    - (zusätzlich eingebauter erweiterter) Hauptspeicher.

Rechnerarchitektur – Peripherie

- Veränderbarkeit:
  - veränderlich:
    - RAM-Speicher (RAM = 'Random-Access Memory', dt. Speicher mit wahlfreiem/beliebigem Zugriff):
      - SRAM: Static RAM (Einsatz als Register- und Cache-Speicher);
      - DRAM: Dynamic RAM (Einsatz als Hauptspeicher).
    - HD/SSD, DVD/BD.
  - unveränderlich:
    - ROM-Speicher (ROM = 'Read-Only Memory', dt. Nurlese-Speicher);
    - Barcode-, RFID-, QR-Tag-Speicher.

Rechnerarchitektur – Vergleich

- Register vs. Cache:

Kriterium	Register	Cache
Geschwindigkeit (in % des vollen CPU-Takts)	100%	10%–50%
Kapazität	max. 1 KB	max. 16 MB (steigend)
Adressierung	über Namen/Symbole	über Adressen/Zahlen
Zugreifbarkeit	direkt lesbar und beschreibbar	nur indirekt via Hauptspeicher lesbar und beschreibbar
Datenunabhängigkeit	Daten vollständig unabhängig vom Hauptspeicher	Daten nur als automatische Spiegelung des Hauptspeichers

Rechnerarchitektur – Vergleich

- HD vs. SSD:

Kriterium	HD	SSD
Geschwindigkeit	100%	250–500%
Kapazität	100%	10–25%
Speicherung	magnetisch mit beweglichen Teilen	elektronisch ohne bewegliche Teile
Zugreifbarkeit	beliebig oft lesbar und beschreibbar	beliebig oft auslesbar, nur 2500–10000 Mal beschreibbar pro Speicherzelle (!)
Zuverlässigkeit	höhere mechanische, niedrigere elektronische Ausfallrate	niedrigere mechanische, höhere elektronische Ausfallrate

Rechnerarchitektur – Leistung

- Die Gesamtleistung eines Rechnersystems hängt von mehreren Faktoren ab:
  - Hardware:
    - Zentraleinheit: Optimierung physikalischer und konzeptioneller Parameter (z. B. Parallelitätsgrad, Zusammenspiel einzelner Komponenten wie CPU und Speicher);
    - Peripherie: Optimierung von Schnittstellen (z. B. USB 1/2/3).
  - Software:
    - Betriebssystem: Optimierung im Hinblick auf die Rechner-architektur (vgl. Linux vs. Windows);
    - Anwendung: Optimierung bezüglich Rechnerarchitektur und Betriebssystem.

Rechnerarchitektur – Leistung

- Die Leistung der Zentraleinheit wird durch zwei grundlegende Faktoren bestimmt:
  - physikalische Parameter:
    - Taktfrequenz: Erhöhung des Taktes (mehr Befehle/sec);
    - Zusammenspiel von Komponenten: Verringerung von Wartezyklen auf langsamere Komponenten (z. B. durch Pufferspeicher wie Caches).
  - konzeptionelle Parameter:
    - Verwaltung: Trennung von Daten und Instruktionen;
    - Parallelitätsgrad: Erhöhung der Parallelverarbeitung von Daten auf verschiedenen Hardware-Ebenen.

Rechnerarchitektur – Leistung

- Beispiel Erhöhung des Parallelitätsgrades:
  - Erhöhung der Anzahl elementarer Operationen, die in einem Takt von einem komplexen Befehl ausgeführt werden können (z. B. N einfache Befehle mit je 1 Takt Dauer zu einem komplexen Befehl mit 1 Takt bündeln [CISC statt RISC]).
  - Erhöhung der Anzahl von Bits/Bytes, die auf einmal von einem Befehl verarbeitet werden können:
    - Datenbreite: Übergang von 8 auf 16, 16 auf 32, 32 auf 64 Bits usw. (32- vs. 64-Bit-Prozessor);
    - SIMD ('Single Instruction Multiple Data'): Verarbeitung mehrerer Datenpakete durch einen einzigen Befehl (z. B. Addition von vier Mal zwei Werten auf einmal statt nur ein Mal zwei Werten).

Rechnerarchitektur – Leistung

- Erhöhung der Anzahl Stufen in der Verarbeitungs-pipeline, so dass pro Takt mehr Instruktionen über-lappend-parallel ausgeführt werden können.
- Erhöhung der Anzahl Ausführungseinheiten pro Rechner:
  - gleichartige Verarbeitungseinheiten:
    - mehr Threads pro Kern (Multi-/Hyperthreading-Architektur);
    - mehr Kerne pro Prozessor (Multicore-Architektur);
    - mehr Prozessoren pro Platine ('Mainboard');
  - ungleichartige Verarbeitungseinheiten (Koprozessoren):
    - Fließkomma-Prozessoren (heute durchweg auf der CPU);
    - Grafik-Prozessoren (meist noch eigene GPU, mit zunehmender Tendenz in die CPU integriert).

Rechnerarchitektur – Leistung

- Erhöhung der Anzahl Rechner, die zusammen-arbeiten:
  - mehrere Platinen/Rechner in einem Cluster;
  - mehrere Cluster in einem 'Schrank';
  - mehrere Schränke in einem (Super-)Computer.
- Erhöhung der Anzahl (Super-)Computer/PCs, die über Netzwerke verbunden sind ('verteiltes Rechnen'):
  - mehrere (Super-)Computer oder PCs in einem internen Netzwerk;
  - mehrere interne (Firmen-)Netzwerke in einem externen Netzwerk (z. B. über Internet: SETI@Home).

Rechnerarchitektur – Leistung

- Durchsatz von Operationen/Anweisungen:
  - unspezifischer Durchsatz (verschiedene Arten von Operationen/Befehlen):
    - IPS = 'Instructions per Second':
      - üblich nur MIPS = 'Million/Mega Instructions per Second';
      - aber nicht GIPS/TIPS/etc., sondern 1000/1000000/etc. MIPS!
    - Beispiele für Rechenleistungen in MIPS:
      - PC-Prozessoren 2010 (Intel Core i7 4 Kerne): ca. 150 000 MIPS für alle Kerne;
      - Smartphone-Prozessoren 2010 (ARM Cortex A9 Dualcore): ca. 5000 MIPS für alle Kerne.

Rechnerarchitektur – Leistung

- spezifischer Durchsatz bei Rechenoperationen:
  - FLOPS = 'Floating Point Operations per Second':
    - 'Floating Point Number' = Fließkomma- oder Gleitpunktzahl = Bruchzahl mit Exponent (wissenschaftliche Darstellung wie auf Taschenrechner);
    - MFLOPS/GFLOPS/TFLOPS/PFLOPS/etc. = Mega-/Giga-/Tera-/Peta-/etc. FLOPS (Basis 10).
  - Beispiele für aktuelle Rechenleistungen in FLOPS:
    - Hauptprozessor PC (CPU): bis zu 150 GFLOPS, Grafikprozessor PC (GPU): bis zu 2500 GFLOPS;
    - schnellster Supercomputer derzeit (5/2016): 33863 TFLOPS = 33.863 PFLOPS (Billiarden Gleitpunkt-Operationen pro Sekunde) bei einer Leistungsaufnahme von 17808 Kilowatt.

Rechnerarchitektur – Leistung

- Durchsatz im Bereich standardisierter, anwendungsbezogener Tests (Benchmarks):
  - Grafikprozessoren (GPU = 'Graphics Processing Unit'):
    - 3DMark (Futuremark): 3D-Leistung von Grafikkarte und PC als Gesamtsystem aus GPU und CPU in Punkten;
    - Cinebench (Maxon): plattformübergreifende Test-Suite für GPU und CPU in Sekunden (basierend auf der 3D-Software CINEMA 4D);
    - Frame-Rate (Frames/s, fps = 'frames per second'): Anzahl berechneter Bilder pro Sekunde (z. B. in bestimmten Computerspielen unter bestimmten Randbedingungen).

Rechnerarchitektur – Leistung

- Hauptprozessoren (CPU = 'Central Processing Unit'):
  - SPEC ('Standard Performance Evaluation Corporation'):
    - SPECint: Ganzzahl-Leistung eines Rechners ('Integer');
    - SPECfp: Bruchzahl-Leistung eines Rechners ('Floating Point').
  - LINPACK: Messung der Leistung eines Supercomputers der Top500-Liste in FLOPS;
  - BAPCo ('Business Applications Performance Corporation'), basierend auf realen Applikationen:
    - WebMark: Benchmark für Internet-Technologien;
    - SYSmark: Benchmark für PC-Performance.
  - PCMark (Futuremark): Gesamtleistung eines PCs in Punkten oder Durchsatz in KByte/s (je nach Test).

Rechnerarchitektur – Leistung

- Größendimensionen von Chipstrukturen:
  - Herstellungsprozess (seit 2000):

Jahr (ca.)	Prozess (Verkleinerungsfaktor)
2000	130 nm = 0.130 µm
2002	90 nm = 0.090 µm
2006	65 nm = 0.065 µm
2008	45 nm = 0.045 µm
2010	32 nm = 0.032 µm
2011	22 nm = 0.022 µm
2013	14 nm = 0.014 µm (geschätzt)
2016	10 nm = 0.010 µm (geschätzt)

÷√2 ÷2

Rechnerarchitektur – Leistung

- Größere und kleinere Strukturen werden wiederum durch Vorsätze ausgedrückt (ausschließlich Basis 10, nicht 2!):

Vorsatz	Größe	Beispiel (ungefähre Größenordnung)
milli (m)	10^{-3}	Druckerpixel $2 \cdot 10^{-3}$ (1200 dpi)
mikro (µ)	10^{-6}	Feinstaub ($10 \cdot 10^{-6}$), menschliches Haar ($50 \cdot 10^{-6}$)
nano (n)	10^{-9}	Mikrobe ($30 \cdot 10^{-9}$), Virus ($10 \cdot 10^{-9}$) Strukturbreite Computerchip ($10 \cdot 10^{-9}$–$30 \cdot 10^{-9}$)
pico (p)	10^{-12}	Atom ($100 \cdot 10^{-12}$)
femto (f)	10^{-15}	Atomkern ($10 \cdot 10^{-15}$), Proton/Neutron ($1 \cdot 10^{-15}$)
atto (a)	10^{-18}	?
zepto (z)	10^{-21}	Wellenlänge Gammastrahlung
yokto (y)	10^{-24}	?

Rechnerarchitektur – Leistung

- Integrationsdichte von Computerchips:
  - Mit zunehmender Miniaturisierung der Bauteile können mehr Schaltelemente (Transistoren) pro 1 cm^2 (100 mm^2) Chipfläche untergebracht werden;
  - Beispiele im 32-nm-Herstellungsprozess für verschiedene Arten von Chips:
    - 2-GBit-Speicherbaustein: ca. 3 Mrd. Transistoren auf 35 mm^2 Fläche (ca. 8600 Millionen Transistoren pro cm^2);
    - Intel Core i7-3960X (6 nutzbare von 8 vorhandenen Kernen): ca. 2.27 Mrd. Transistoren auf 435 mm^2 Fläche (ca. 522 Millionen Transistoren pro cm^2).

Rechnerarchitektur – Leistung

- Maße für die Geschwindigkeit von Computer-chips betreffen die Taktfrequenz und Zugriffs-effizienz von Chips:
  - Taktfrequenz: Anzahl der Schaltvorgänge pro Zeiteinheit (minimale Dauer der Änderung des logischen Zustands eines Chips);
  - Zugriffseffizienz: Dauer/Verzögerung des Zugriffs auf den Chip (bzw. zeitlicher Mindestabstand zwischen zwei aufeinanderfolgenden Zugriffen).

Rechnerarchitektur – Leistung

- Taktfrequenz von Computerchips:
  - Die Taktfrequenz wird in Hertz (Einheit Hz) = 1/s = s^{-1} gemessen (Schwingungen/Zyklen pro Sekunde).
  - Aktuelle Prozessor-Geschwindigkeiten bewegen sich im GHz-Bereich:

Taktfrequenz	Schaltzeit	Lichtstrecke (299792.5 km/s) pro Takteinheit
1 GHz = $1 \cdot 10^9$ Hz	1.00 ns = $1.00 \cdot 10^{-9}$ s	~ 0.300 m = 30 cm
2 GHz = $2 \cdot 10^9$ Hz	0.50 ns = $0.50 \cdot 10^{-9}$ s	~ 0.150 m = 15 cm
3 GHz = $3 \cdot 10^9$ Hz	0.33 ns = $0.33 \cdot 10^{-9}$ s	~ 0.100 m = 10 cm
4 GHz = $4 \cdot 10^9$ Hz	0.25 ns = $0.25 \cdot 10^{-9}$ s	~ 0.075 m = 7.5 cm

Rechnerarchitektur – Leistung

- Zugriffseffizienz von Computerchips:
  - Die Zugriffseffizienz spielt v. a. beim wahlfreien Zugriff auf Speicherchips (beliebig angewählte Dateneinheit wie Byte/Wort) eine wichtige Rolle;
  - typische Zugriffsdauer bei verschiedenen Chips:
    - Cache-Speicher (schnellere Speicherbausteine SRAMs): ca. 1–5 ns (schnellster L1-Cache im Prozessor);
    - Hauptspeicher (langsamere Speicherbausteine DRAMs): ca. 10–50 ns;
    - Flash-Speicher (Flash-Chips): ca. 0.25–1 ms (Festplatten im Bereich von 10 ms).

Kontrollfragen

Frage	Antwort
Nennen Sie ein Beispiel aus der realen Welt, wo eine Fließband-Verarbeitung ('Pipelining') ins Stocken gerät: • Was bringt analog eine CPU-Pipeline ins Stocken? • Was könnte man dagegen tun?	
Wie weit kommt das Licht in 1 Taktzyklus bei einer Taktfrequenz von 2.5 bzw. 5 GHz? (Licht-geschwindigkeit c = 299792458 m/s).	

Kontrollfragen

Frage	Antwort
In einer Pipeline mit 12 Stufen dauert es 12 Takte, bis ein Befehl vollständig abgearbeitet wurde: • Wie lange braucht ein Befehl bei 2 GHz Takt, bis er komplett ausgeführt wurde ('Latenz', Verzögerung)? • Wie viele Befehle pro Sekunde kann ein mit 4 GHz getakteter Prozessor in einer 12-stufigen Pipeline im Idealfall maximal abarbeiten?	

Kontrollfragen

Frage	Antwort
Warum wird die Rechenleistung von Supercomputern eher in FLOPS (Fließkomma-Operationen) anstelle von MIPS (Ganzzahl-Operationen) gemessen?	
Warum ist wohl eine GPU (Grafik-Prozessor) im Bereich FLOPS etwa 10 Mal schneller als eine CPU (Hauptprozessor)? Denken Sie daran, was mit einer GPU bzw. CPU jeweils hauptsächlich berechnet wird.	

Assembler

- Jeder Prozessortyp besitzt eine eigene native (fest-verdrahtete, eingebaute) 'Maschinensprache', in der er ausschließlich programmiert werden kann:
  - Die Maschinensprache ('Maschinenkode') besteht aus einer Menge von 'Low-Level'-Instruktionen, die aus einer Folge von Bits mit bestimmten Bedeutungen bestehen; diese sagen dem Prozessor genau, was er tun soll.
  - Assembler ist eine symbolische Darstellung des Maschinen-kodes mit merkbaren Bezeichnungen anstelle von Bitfolgen, die 1:1 in Maschinenkode übersetzt werden können.
  - Jede höhere Programmiersprache wie Java, JavaScript, C# usw. muss zwingend in Maschinenkode übersetzt werden, um das Hochsprachen-Programm ausführen zu können.

Assembler

- Beispiele:
  - Addition zweier 16-Bit-Werte:

Maschinenkode:	Assembler:	Hochsprache (Java):
`B8 12 34`	`MOV AX,1234H`	`0x1234+0x8765`
`05 87 65`	`ADD AX,8765H`	

  - Herunterzählen von 10 bis 0:

Maschinenkode:	Assembler:	Hochsprache (Java):
`B8 00 0A`	`MOV AX,10`	`short X = 10;`
`48`	`Loop:DEC AX`	`while (X > 0)`
`75 FD`	`JNZ Loop`	`  X--;`

Assembler

- Umsetzung der Formel 1+2•(5−3)/4 in Assemblerkode analog Einzelschritten auf Taschenrechner:

Schritt	Taschenrechner-Aktion	Assemblerkode	Ergebnis in EAX
1	Eingabe '5'	`MOV EAX,5`	5
2	Eingabe '−'		
3	Eingabe '3'	`SUB EAX,3`	2 = 5−3
4	Eingabe '•'		
5	Eingabe '2'	`MUL EAX,2`	4 = 2•2
6	Eingabe '÷'		
7	Eingabe '4'	`DIV EAX,4`	1 = 4÷4
8	Eingabe '+'		
9	Eingabe '1'	`ADD EAX,1`	2 = 1+1

Assembler

Anmerkungen:

- EAX kann man sich als Speicherplatz vorstellen, in dem die aktuellen Zwischen- oder Endergebnisse 'gemerkt' (abgespeichert) werden (analog dem aktuell auf dem Taschenrechner angezeigten Wert, der intern gespeichert werden muss).
- Ein Maschinen-/Assembler-Programm ist letztlich eine gespeicherte formalisierte Bediensequenz ('Programm') für einen (Taschen-)Rechner bzw. dessen Prozessor/ALU als ausführende Recheneinheit, die jederzeit abgerufen und ausgeführt werden kann.

Assembler

- Registersatz der 80x86-Prozessorfamilie (Intel- und AMD-Prozessoren für gängige PCs):
  - Allgemeine Ganzzahl-Register (Mehrzweck-Rechen-Register):
    - Accumulator A (Bezeichnung als Reminiszenz an uralte Rechner):

Bits	63–56	55–48	47–40	39–32	31–24	23–16	15–8	7–0
64	RAX							
32					EAX			
16							AX	
8|8							AH	AL

    - Base-Register B: RBX=[32 Bits|EBX=[16 Bits|BX=[BH|BL]]];
    - Counter-Register C: RCX=[32 Bits|ECX=[16 Bits|CX=[CH|CL]]];
    - Data-Register D: RDX=[32 Bits|EDX=[16 Bits|DX=[DH|DL]]].

Assembler

- Spezielle Ganzzahl-Register (Adress-Indexierung):
  - Source-Index: RSI=[32 Bits|ESI=[16 Bits|SI]];
  - Destination-Index: RDI=[32 Bits|EDI=[16 Bits|DI]];
  - Base-Pointer: RBP=[32 Bits|EBP=[16 Bits|BP]];
  - Stack-Pointer: RSP=[32 Bits|ESP=[16 Bits|SP]].
- Segment-Register (Zusatzbits für Adressberechnung):
  - Code-Segment: CS=[16 Bits];
  - Data-Segment: DS=[16 Bits];
  - Stack-Segment: SS=[16 Bits];
  - Extra-Segment: ES=[16 Bits].

Assembler

- Zusatz-Register im 64-Bit-Modus:
  - Mehrzweckregister: R08 bis R15
    (zusätzliche Ganzzahl-Register für allgemeine Rechen-aufgaben);
  - Segment-Register: FS=[16 Bits], GS=[16 Bits]
    (zusätzliche Datensegmente für bestimmte Zwecke).
- Register für Ablaufsteuerung:
  - Instruction-Pointer (IP):
    RIP=[32 Bits|EIP=[16 Bits|IP]];
  - Status-Register der ALU:
    RFLAGS=[32 Bits|EFLAGS=[16 Bits|FLAGS]].

Assembler

Überblick über die wichtigsten Bits des Statusregisters der ALU:

Bit	Name	Beschreibung
0	CF ('Carry Flag')	1, wenn dualer Übertrag/Untertrag entsteht (+)
1	–	(ungenutzt)
2	PF ('Parity Flag')	1, wenn Anzahl Bits des untersten Bytes gerade
3	–	(ungenutzt)
4	AF ('Auxiliary CF')	1, wenn dezimaler Über-/Untertrag entsteht
5	–	(ungenutzt)
6	ZF ('Zero Flag')	1, wenn Ergebnis der letzten Operation 0 ist
7	SF ('Sign Flag')	1, wenn linkestes/höchstes Bit einer Zahl 1 ist
8	TF ('Trace Flag')	1, wenn Einzelschrittmodus für Debugging
9	IF ('Interrupt Flag')	1, wenn aktuelles Programm unterbrechbar
10	DF ('Direction Flag')	1, wenn Zeichenketten absteigend verarbeitet
11	OF ('Overflow Flag')	1, wenn dualer Über-/Untertrag entsteht (±)
12–15	–	(ungenutzt)

Assembler – Programmierung

- Grundlegendes Befehlsformat der 80x86-Prozessoren:
  - Befehle werden nach Anzahl Operanden unterschieden:

Operanden	Befehlsformat	Beispiel
2 (Ziel und/ oder Quelle[n])	`Befehlsname Ziel,Quelle` `Befehlsname Quelle`$_1$`,Quelle`$_2$	ADD EAX,ECX (Addition) CMP EAX,ECX (Komparation)

1 (Ziel oder Quelle)	`Befehlsname Ziel`
	
	`Befehlsname Quelle`
		NOT EAX (Negation)
		MUL ECX (Multiplikation)

0	`Befehlsname`	NOP (Noperation)

Assembler – Programmierung

- `Ziel` und `Quelle` können verschiedener Art sein:
  - *Register*: EAX, EBX usw., AX, BX etc., AH/AL, BH/BL usw. (verarbeitet wird der Inhalt/Wert des Registers);
  - *Direktwert* (Konstante): 12345678 (dezimal, ohne weitere Kennzeichnung), 1234ABCDh (hexadezimal, kenntlich gemacht durch ein nachgestelltes 'h');
  - *Adresse*:
    - [EBX], [ESI] usw. (verarbeitet wird nicht der Inhalt des Registers, sondern der Wert an der Speicheradresse, die das Register EBX, ESI usw. enthält);
    - [P], [XYZ] usw. (verarbeitet wird der Wert an der Adresse, für den der Variablenname P, XYZ usw. steht).

Assembler – Programmierung

- Werte in `Ziel` werden geschrieben und damit verändert (Ergebnis einer Operation), Werte in `Quelle` werden nur gelesen und bleiben dabei unverändert.
- Fast alle Operationen setzen das Statusregister der ALU, um bestimmte Eigenschaften des Ergebnisses der letzten Operation festzuhalten: z. B. wenn das Ergebnis
  - einen Wert gleich 0 oder ungleich 0 annimmt;
  - einen negativen/positiven Wert hat (im Zweierkomplement);
  - einen Überlauf produziert hat (z. B. bei der Addition zweier 32-Bit-Werte, die nicht mehr in ein 32-Bit-Register passen und somit ein Übertrag auf die 2^{32}-Stelle erzeugt wird).

  Bestimmte Befehle können den letzten Status abfragen und in Abhängigkeit davon den Programmfluss steuern.

Assembler – Programmierung

- Befehlsgruppen für 80x86-Intel- und AMD-Assembler (unvollständig):
  - Transportbefehle (Daten kopieren):
    - Kopieren: `MOV Ziel,Quelle;`
    - Vertauschen: `XCHG` $\texttt{Ziel}_1/\texttt{Quelle}_1,\texttt{Ziel}_2/\texttt{Quelle}_2$.
  - Logikbefehle (logische Verknüpfungen):
    - Negation (logisch, Einer-Komplement): `NOT Ziel;`
    - Konjunktion: `AND Ziel,Quelle;`
    - Disjunktion: `OR Ziel,Quelle;`
    - Kontrajunktion: `XOR Ziel,Quelle.`

Assembler – Programmierung

- Arithmetikbefehle (arithmetische Verknüpfungen):
  - Addition: `ADD Ziel,Quelle;`
  - Subtraktion: `SUB Ziel,Quelle;`
  - Multiplikation: `MUL [Ziel,]Quelle`
    (32 Bits in EAX werden mit 32 Bits der Quelle multipliziert; das Ergebnis besteht aus 64 Bits in EDX und EAX [d. h. die höherwertigen 32 Bits 2^{63}–2^{32} werden in EDX, die niederwertigen 32 Bits 2^{31}–2^{0} in EAX gespeichert]);
  - Division: `DIV [Ziel,]Quelle`
    (64 Bits in EDX und EAX werden durch 32 Bits der Quelle dividiert, wobei der 32-Bit-Quotient in EAX, der 32-Bit-Divisionsrest in EDX landet).

Assembler – Programmierung

- Inkrementierung: `INC Ziel` (Erhöhung um 1);
- Dekrementierung: `DEC Ziel` (Erniedrigung um 1);
- Negation (arithmetisch, Zweier-Komplement): `NEG Ziel` (Berechnung von –Ziel aus +Ziel durch 0–Ziel);
- Bitverschiebung links: `SHL Ziel,1` (Verschieben aller Bits um eine Stelle nach links, wobei das linkeste Bit herausfällt und das rechteste Bit eine 0 erhält; Beispiel für 8 Bits: 10011001 wird zu 00110010);
- Bitverschiebung links: `SHR Ziel,1` (Verschieben aller Bits um eine Stelle nach rechts, wobei das rechteste Bit herausfällt und das linkeste Bit eine 0 erhält; Beispiel für 8 Bits: 10011001 wird zu 01001100).

Assembler – Programmierung

- Vergleichsbefehle:
  - Testen: `Test` $\texttt{Quelle}_1$,$\texttt{Quelle}_2$
    (implizit logische Konjunktion der beiden Operanden, um das Statusregister zu setzen; die beiden Operanden $\texttt{Quelle}_1$ und $\texttt{Quelle}_2$ bleiben dabei jedoch unverändert);
  - Vergleichen (Komparation): `CMP` $\texttt{Quelle}_1$,$\texttt{Quelle}_2$
    (implizite arithmetische Subtraktion der beiden Operanden, um das Statusregister zu setzen; die beiden Operanden $\texttt{Quelle}_1$ und $\texttt{Quelle}_2$ bleiben dabei jedoch unverändert).

Assembler – Programmierung

- Programmsteuerung:
  - unbedingte Verzweigung (Sprung zu anderem Befehl ohne Bedingung): `JMP Zieladresse;`
  - bedingte Verzweigung (Sprung zu anderem Befehl unter einer bestimmten Bedingung): `JX Zieladresse;`
    X ist ein Platzhalter für eine konkret einzusetzende Vergleichsbedingung:

Kode X	Beschreibung: verzweige, wenn	Beispiel
e bzw. z	gleich ('equal') bzw. null ('zero')	`je Adr`
ne bzw. nz	nicht gleich ('not equal') bzw. nicht null ('not zero')	`jnz Adr`
l(e)	kleiner als (oder gleich) ('less than [or equal]')	`jl Adr`
g(e)	größer als (oder gleich) ('greater than [or equal]')	`jge Adr`

Assembler – Programmierung

- Statt einer Zieladresse in Form einer Zahl für eine Speicherstelle wird jedoch grundsätzlich ein Label (Sprungmarke, Tag) angegeben, das denjenigen Befehl kennzeichnet, zu dem verzweigt werden soll (dem Label folgt meist ein Doppelpunkt zur Abgrenzung vom Befehl):

```
            JMP Label
            …
Label:      Befehl ; Ich bin nur ein Kommentar!
```

- Zur Erläuterung oder Dokumentation des Programms kann hinter jedem Befehl ein Kommentar gesetzt werden, der durch einen Strichpunkt vom Befehl abgetrennt wird (reine Kommentarzeilen möglich).

Assembler – Programmierung

- Beispielprogramme:
  - Berechnung der Summe aller Zahlen von 1 bis N ($\Sigma_{K=1..N}$ K) nach EAX (bzw. rückwärts von N bis 1, was zum selben Ergebnis führt):

```
; Berechnung der Summe von 1 bis N
; in [N] steht der zu berechnende Wert
        MOV ECX,[N]          ; Summe 1 bis N (N bis 1)
        MOV EAX,0            ; Ergebnissumme in EAX
Loop:   ADD EAX,ECX          ; nächsten Wert auf Ergebnis
        DEC ECX              ; addieren und herunterzählen
        JNZ Loop             ; bis 0 (dann kein Sprung!)
        ; in EAX steht schließlich Summe von 1 bis N
        …
```

Assembler – Programmierung

Ablauf des Programms für N = 5 (Z ist ein Statusbit des Statusregisters, das anzeigt, ob das Ergebnis der letzten Operation Null [Z = 1] oder Nicht-Null [Z = 0] war):

Schritt	Assembler-/Maschinen-Befehl	EAX	ECX	Statusregister
0	…	?	?	Z = ?
1	`MOV ECX,[N]`	?	5	Z = 0
2	`MOV EAX,0`	0	5	Z = 1
3	`ADD EAX,ECX`	5	5	Z = 0
4	`DEC ECX`	5	4	Z = 0
5	`JNZ Loop`	5	4	Z = 0
6	`ADD EAX,ECX`	9	4	Z = 0
7	`DEC ECX`	9	3	Z = 0

Assembler – Programmierung

Schritt	Assembler-/Maschinen-Befehl	EAX	ECX	Statusregister
8	JNZ Loop	9	3	Z = 0
9	ADD EAX,ECX	12	3	Z = 0
10	DEC ECX	12	2	Z = 0
11	JNZ Loop	12	2	Z = 0
12	ADD EAX,ECX	14	2	Z = 0
13	DEC ECX	14	1	Z = 0
14	JNZ Loop	14	1	Z = 0
15	ADD EAX,ECX	15	1	Z = 0
16	DEC ECX	15	0	Z = 1
17	JNZ Loop	15	0	Z = 0
Ende des Programms: in EAX steht Endergebnis 15				

Assembler – Programmierung

- Berechnung der größeren Zahl (Maximum) zweier gegebener Zahlen X und Y nach EAX:

```
; Berechnung des Maximums zweier Zahlen X und Y
; Wert von X an Speicheradresse [X]
; Wert von Y an Speicheradresse [Y]
        MOV EAX,[X]          ; X in Register EAX laden
        MOV EDX,[Y]          ; Y in Register EDX laden
        CMP EAX,EDX          ; Zahl 1 verglichen mit Zahl 2
        JG  Done             ; ok, wenn X (EAX) > Y (EDX)
        MOV EAX,EDX          ; sonst    X (EAX) ≤ Y (EDX)
Done:   ; fertig: größere Zahl steht jetzt in EAX
        …
```

Assembler – Programmierung

- Test, ob eine Ganzzahl gerade (d. h. durch 2 teilbar) ist oder nicht:

```
; Berechnung der Geradheit einer Zahl Z (Geradheit
; entspricht restloser Teilbarkeit durch 2)
; Zahl Z an Speicheradresse [Z]
      MOV EAX,[Z]          ; Zahl Z laden, alle Bits bis
      AND EAX,1            ; auf unterstes 2⁰ löschen
      JNZ Odd              ; wenn noch 1, dann ungerade
      … ; Zahl ist gerade (tue irgendwas hier und
      JMP Cont             ; springe dann zu Cont[inue])
Odd:  … ; Zahl ist ungerade (tue irgendwas hier)
Cont: … ; hier geht normales Programm weiter
```

Übungsaufgaben

- Welchen Effekt haben folgende Assemblerbefehle:

Befehl	Effekt
`XCHG EAX,EAX`	
`SUB EAX,EAX`	
`ADD EAX,EAX`	
`MUL EAX,EAX`	
`DIV EAX,EAX`	
`XOR EAX,EAX`	
`AND EAX,EAX`	
`OR EAX,EAX`	

Übungsaufgaben

- Was ist der Unterschied zwischen folgenden beiden Befehlspaaren:

Befehl 1	Befehl 2	Unterschied
`TEST EAX,EBX`	`AND EAX,EBX`	
`CMP EAX,EBX`	`SUB EAX,EBX`	

- Prüfen und begründen Sie, ob folgende Befehle gleiche oder verschiedene Ergebnisse liefern:

Befehl 1	Befehl 2	Ergebnis
`SHL EAX,1`	`ADD EAX,EAX`	
`SHR EAX,1`	`SUB EAX,EAX`	

Übungsaufgaben

- Analysieren Sie folgende Assemblerprogramme auf ihre Funktionalität und ermitteln Sie, was das jeweilige Programm als Ergebnis berechnet:

```
; Klausuraufgabe WS 11/12
        MOV ECX,10              ; initialisiere ECX mit 10
        MOV EBX,0               ; initialisiere EBX mit 0
Loop:   MOV EAX,ECX             ; kopiere ECX nach EAX
        MUL EAX                 ; multipliz. EAX mit EAX nach EAX
        ADD EBX,EAX             ; addiere EAX auf EBX
        DEC ECX                 ; erniedrige ECX um 1
        JNZ Loop                ; springe zurück zur Stelle Loop,
                                ; wenn letztes Ergebnis ≠ 0
        ; Endergebnis steht in EBX
```

Übungsaufgaben

```
; Assembler-Programm für bestimmte Funktion mit
; zwei Variablen X und Y
        MOV EAX,[X]                 ; X nach EAX kopieren
        AND EAX,[Y]                 ; Y dazukonjungieren
        NOT EAX                     ; Ergebnis negieren
        ; Endergebnis in EAX
```

```
; Assembler-Programm für bestimmte Funktion mit
; zwei Variablen X und Y
        MOV EAX,[X]                 ; X nach EAX kopieren
        OR  EAX,[Y]                 ; Y dazudisjungieren
        NOT EAX                     ; Ergebnis negieren
        ; Endergebnis in EAX
```

Übungsaufgaben

```
; Assembler-Programm mit zwei verschachtelten Schleifen
       MOV ECX,10          ; initialisiere ECX mit 10
Loop1: MOV EBX,10          ; initialisiere EBX mit 10
Loop2: MOV EAX,ECX         ; kopiere ECX nach EAX
       MUL EBX             ; multipliz. EAX mit EBX nach EAX
       ; in EAX steht nun ein Berechnungsergebnis, das
       ; z. B. auf dem Bildschirm ausgegeben werden kann
       DEC EBX             ; erniedrige EBX um 1
       JNZ Loop2           ; springe zu Loop2, wenn EBX ≠ 0
       DEC ECX             ; erniedrige ECX um 1
       JNZ Loop1           ; springe zu Loop1, wenn ECX ≠ 0
       ; hier endet das Programm (es wird kein Endergebnis
       ; berechnet, sondern die Zwischenergebnisse in EAX
       ; oben werden ausgegeben oder anderweitig
       ; weiterverarbeitet)
```

Übungsaufgaben

```
; Manipulation von Bitgruppen
        MOV EAX,12345678h           ; Beispielzahl
        MOV BX,AX
        SHL EBX,16
        SHR EAX,16
        OR EAX,EBX
        ; Was steht jetzt in EAX? Was tut das Programm?
```

```
; Klausuraufgabe SS 12
        MOV AL,10                   ; Zahl 1
        MOV DL,20                   ; Zahl 2
        XOR AL,DL                   ; die Zahlen 1 und 2
        XOR DL,AL                   ; mehrmals kontra-
        XOR AL,DL                   ; jungieren
Done:   ; Was steht in AL und DL? Was tut das Programm?
```

Übungsaufgaben

- Schreiben Sie Assembler-Programme für folgende Aufgaben:
  - Schwierigkeitsgrad leicht:
    - Austausch der beiden Registerinhalte von EAX und EDX, ohne den XCHG-Befehl zu benutzen;
    - Berechnung der Subjunktion und Bi(sub)junktion zweier Werte X und Y (Hinweis: $X \rightarrow Y = \neg X \vee Y$);
    - Mittelwertberechnung von 4 Zahlen in EAX, EBX, ECX und EDX nach EAX;
    - Berechnung des Minimums zweier Zahlen nach EAX analog zum Maximum zweier Zahlen;
    - Berechnung des mathematischen Ausdrucks $2 \bullet (7-4) \div 3+8$ nach EAX.

Übungsaufgaben

- Schwierigkeitsgrad mittel:
  - Test auf Teilbarkeit einer Ganzzahl durch 4 (8, 16 usw.) analog Test auf Geradheit bzw. Teilbarkeit durch 2 (ohne den Divisionsbefehl mit Restwertbildung zu benutzen!).
  - Ermittlung des Absolutbetrags einer Zahl Z im Zweierkomplement (ABS-Funktion):
    - ABS(Z) = −Z, wenn Z < 0;
    - ABS(Z) = Z, wenn Z ≥ 0.
  - Ermittlung des Vorzeichens einer Zahl Z im Zweierkomplement nach EAX (SIGN-Funktion):
    - SIGN(Z) = −1, wenn Z < 0;
    - SIGN(Z) = 0, wenn Z = 0;
    - SIGN(Z) = +1, wenn Z > 0.

Übungsaufgaben

- Berechnung des Produkts der Zahlen von 1 bis N ('Fakultät') analog zur Summe aller Zahlen von 1 bis N:
  - Fakultät allgemein: $N! = N \bullet (N-1) \bullet \ldots \bullet 1$;
  - Sonderfälle der Fakultät: $0! = 1$, $1! = 1$.
- Schwierigkeitsgrad schwer:
  - Berechnung des Wertes 3^N nach EAX, wobei N eine Variable mit einem Wert ≥ 0 an einem entsprechenden Speicherplatz sein soll;
  - Berechnung des Mittelwerts von N Zahlen ab Speicherstelle A nach EAX;
  - Umkehrung der Reihenfolge aller Bits einer 32-Bit-Zahl in EDX nach EAX.