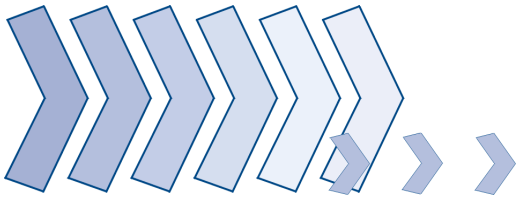
HUMBOLDT-UNIVERSITÄT ZU BERLIN



Digitale Systeme
Instruktionssatzarchitekturen

Dr.-Ing. Siegmar Sommer

Sommersemester 2023

Einführung

- Bisher behandelt:
  - die prinzipielle Idee von Instruktionssatzarchitekturen, Maschineninstruktionen und ihre Codierung
  - einige der Freiheitsgrade beim Entwurf einer Architektur (RISC/CISC, Endianness,…)
- Jetzt: ein etwas erweiterter Blick auf Befehlssätze und die Möglichkeiten, eine „programmierbare Maschine“ zu gestalten

Befehlsformate

- Befehlsformat = Länge und Struktur eines Maschinenbefehls
- Faktoren für den Entwurf von Befehlsformaten:
  - Anzahl der Befehle und ihre Flexibilität (Operanden)
  - Adressierung und Adressierungsarten
  - Aufbau (Aufwand für die Decodierung)
  - feste oder variable Länge der Instruktionen
- Traditionell: Kosten der Hardware zum Dekodieren und Ausführen der Befehle (RISC vs. CISC) spielen eine Rolle

Befehlssätze

- Befehlssatz: Gesamtheit der zur Verfügung stehenden Befehle
- Der Befehlssatz ist einer der wichtigsten Faktoren der Rechnerarchitektur:
  - bestimmt die Komplexität der Programme
  - bestimmt die Komplexität von Schaltungen und Kosten

Einfachheit des Programmierens

vs.

Prozessorkomplexität

Beispiele

Computer/Prozessor	Befehle	RISC/CISC
CDC 3600	124	RISC
IBM 360/370; IBM z10	220-256 ; 894	CISC
PDP-11	64	CISC
M6800	84	CISC
Alpha	Über 200	RISC
Vax 11 / 750	304	CISC
Motorola 68000	über 130	CISC
IA32	über 200	CISC
HP-PA	170	RISC
MIPS	40	RISC

Beispiele für mögliche Befehlsklassen

- Es gibt typische „Klassen“ von Instruktionen, die sich (in jeweils etwas unterschiedlichen Varianten) in fast allen Instruktionssätzen wiederfinden
  - Datenübertragung (MOVE/STORE/FETCH, PUSH+POP,…)
  - Arithmetik (ADD, SUB, INC, SHIFT,…)
  - Logik (AND, OR, XOR, EQU,…)
  - Ein- und Ausgabe (READ, WRITE,…)
  - Programmablaufsteuerung (JUMP, bedingtes JUMP, CALL und RET, TEST, NOP,…)

Grundbefehlssatz

Typ	Operation	Beschreibung
Datenübertragung	Move (Copy)	Kopieren eines Wortes oder Blockes von der Quelle zum Ziel
	Store	Kopieren eines Wortes vom Prozessor zum externen Speicher.
	Load (Fetch)	Kopieren eines Worts vom ext. Speicher zur CPU
	Exchange	Austausch der Inhalte von Quelle und Ziel.
	Clear (Reset)	Transfer eines Wortes bestehend aus Nullen zum Ziel.
	Set	Transfer eines Wortes bestehend aus Einsen zum Ziel.
	Push	Kopieren eines Wortes von der Quelle zum obersten Kellerspeicherplatz (Top of Stack)
	Pop	Kopieren eines Wortes vom obersten Kellerspeicherplatz (Top of Stack) zum Ziel.

Grundbefehlssatz

Typ	Operation	Beschreibung
Datenmani-pulation	Shift	Links- (Rechts-) Schieben des Operanden
	Rotate	Links- (Rechts-) Schieben des Operanden auf einem geschlossenen Pfad.
	Convert (Edit)	Änderung des Datenformates, z.B.: Binär- nach BCD-Repräsentation.
Arithmetisch	Add	Berechnung der Summe von zwei Operanden.
	Subtract	Berechnung der Differenz von zwei Operanden.
	Multiply	Berechnung des Produktes von zwei Operanden.
	Divide	Berechnung des Quotienten von zwei Operanden
	Absolute	Ersetze den Operanden durch den Absolutwert
	Negate	Ändere das Vorzeichen des Operanden
	Increment	Addiere 1 zum Operanden
	Decrement	Subtrahiere 1 vom Operanden

Grundbefehlssatz

Typ	Operation	Beschreibung
Logisch	And, Or, Not, Xor, Equivalence	Bitweises Ausführen der logischen Operation
Eingabe- und Ausgabe	Input (Read)	Kopieren von Daten vom angegebenen Ein/Ausgabe Port (d.h. Gerät) zum Ziel; z.B.: Hauptspeicher oder Prozessorregister
	Output (Write)	Kopieren von Daten zum angegebenen Quell-Ein/Ausgabe Port (d.h. Gerät)
	Start IO	Transfer von Befehlen zum Ein/Ausgabe Port, um die Ein/Ausgabe-Operation zu initiieren.
	Test IO	Transfer von Statusinformation vom Ein/Ausgabe-System zum angegebenen Ziel
	Halt IO	Transfer von Befehlen zum Ein/Ausgabe Port, um die Ein/Ausgabe-Operation zu beenden.

Grundbefehlssatz

Typ	Operation	Beschreibung
Programmkontrolle	Jump (Branch)	Führe Programm an angegebener Adresse weiter aus; bedingungsloser Programmsprung; lade IP mit angegebener Adresse
	Jump Conditional	Bedingungstest: In Abhängigkeit von der Bedingung lade entweder IP mit angegebener Adresse oder führe nächstfolgenden Befehl aus
	Jump to Subroutine (Call)	Speichere aktuelle Programmsteuerungsinfor-mation (IP, status register, usw.) an einen bekannten Ort z.B. Top of the Stack, und springe zur angegebenen Adresse
	Return	Ersetze Inhalte des IP, Status Register, usw. mit der Information von dem bekannten Ort, z.B.: vom Top of the Stack
	Execute	Hole den Operanden von dem angegebenen Ort, und führe den Befehl aus; der IP wird nicht verändert.

Grundbefehlssatz

Typ	Operation	Beschreibung
Programmkontrolle	Skip	Erhöhe IP, um den nächsten Befehl zu überspringen
	Skip Conditional	Teste angegebene Bedingung. In Abhängigkeit vom Ergebnis erhöhe IP oder tue nichts.
	Test	Teste angegebene Bedingung. Setze Flag(s) abhängig vom Ergebnis.
	Compare	Führe einen logischen oder arithmetischen Vergleich von zwei oder mehr Operanden durch. Setze Flags, die von dem Ergebnis abhängig sind.
	Set Control Register	Klasse von Befehlen, welche CPU-interne Mechanismen steuern, z.B. für Schutzzwecke: Zeitgeber, Ausführung privilegierte Befehle, Verhinderung von unberechtigten Speicherzugriffen, Fehlererkennung, usw.

Grundbefehlssatz

Typ	Operation	Beschreibung
Programm-kontrolle	Halt	Anhalten der Programmausführung
	Wait (Hold)	Anhalten der Programmausführung. Testet kontinuierlich auf eine angegebene Bedingung. Falls die Bedingung erfüllt wird, dann wird die Abarbeitung wieder aufgenommen.
	No Operation	Es wird nichts getan. Das Programm läuft weiter.

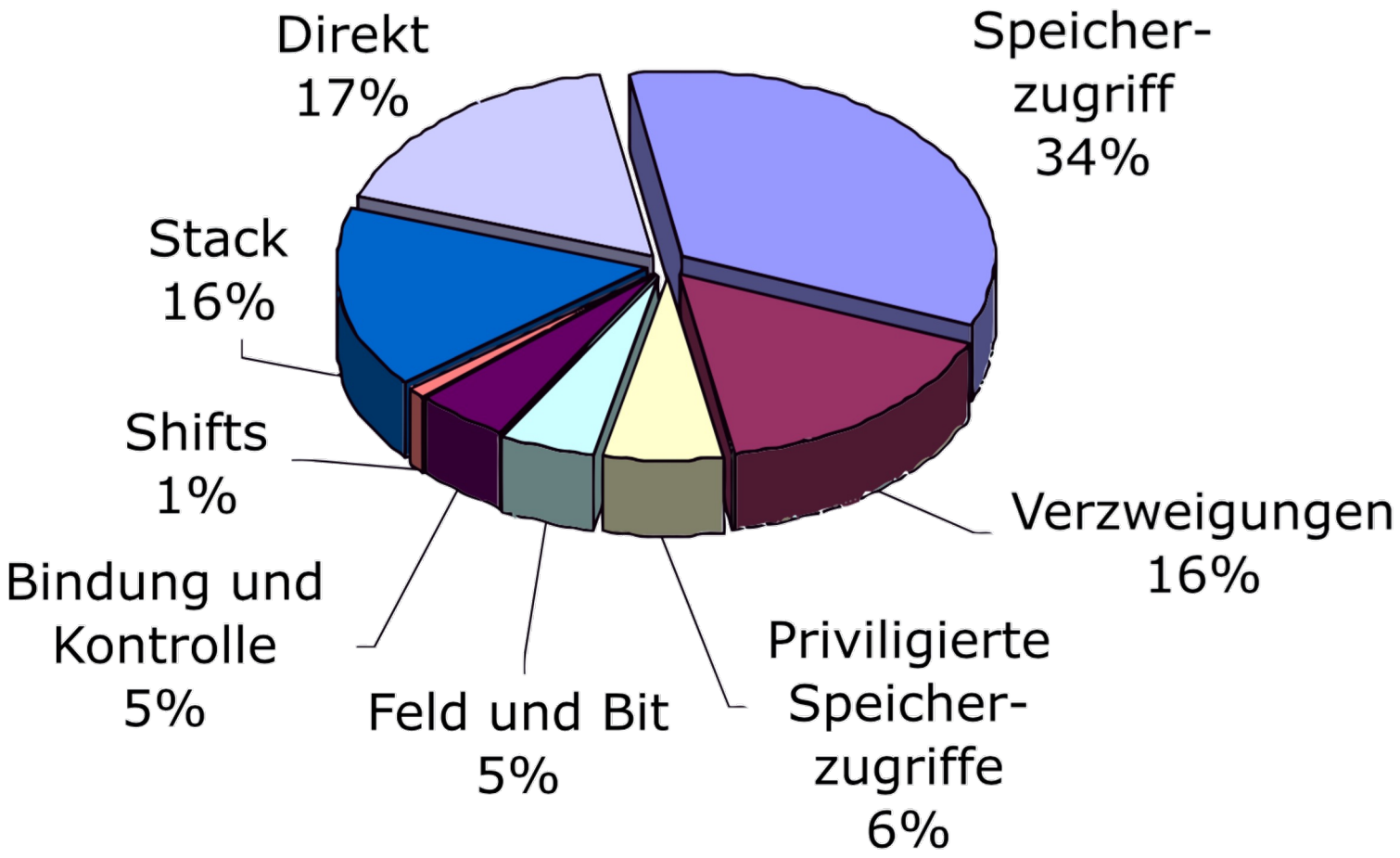
- Die in den Tabellen angegebenen Befehle müssen nicht in jedem Befehlssatz vorhanden sein, sie schließen sich u.U. sogar gegenseitig aus, z.B. kommen meist entweder **load/store**- oder **mov**-Befehle vor

Erweiterung des Befehlsatzes

- Viele Instruktionen kennen erweiterte Befehlssätze für spezielle Aufgaben, wie z.B.
  - Fließkomma-Arithmetik
  - Verarbeitung von Grafik-Daten
  - Verarbeitung von Datenreihen (z.B. bei Signalprozessoren)
  - Mehrstufige Unterprogrammbindung, automatische Indizierung
  - Operationen auf Vektoren und Felder
  - erweiterte Genauigkeiten beim arithmetischen Operationen

  . . .

Beispiel eines Instruktionsmixes



Operanden-Adressierung

- Es gibt wesentliche Unterschiede zwischen den Architekturen hinsichtlich der Anzahl der Operanden, die in den Instruktionen angegeben werden können
- Unterscheidung betrifft speziell die Zahl der Operanden für ALU-Instruktionen
- Im Allgemeinen spricht man bei einer Architektur, deren (ALU-)Instruktionen **n** Operanden enthalten, von einer **n-Adress-Maschine**

3-Adress-Maschine

- 3-Adress-Maschine: Befehle enthalten bis zu drei Operanden bzw. Operandenadressen
- Eine der Adressen adressiert den Ergebnisoperanden
- Oft zu finden in klassischen RISC-Architekturen (z.B. Motorola/IBM PowerPC)
- Ermöglicht sehr kurze Programme
- Für folgende Beispiele gilt:
  - Algorithmus: $X = A*B + C*C$
  - X, A, B, C, T seien Bezeichner für Speicherzellen des Hauptspeichers

Befehl		**Kommentar (Ergebnis: X = A * B + C * C)**
MULT	T, A, B	**T ⟵ A * B**
MULT	X, C, C	**X ⟵ C * C**
ADD	X, T, X	**X ⟵ T + X**

2-Adress-Maschine

- 2-Adress-Maschine: Ein Befehl enthält bis zu zwei Operanden bzw. Operandenadressen
- Einer der Operanden wird durch das Ergebnis überschrieben

Befehl		Kommentar (Ergebnis: **X = A * B + C * C**)
MOV	R0, A	**R0 ← A**
MULT	R0, B	**R0 ← R0 * B**
MOV	X, C	**X ← C**
MULT	X, C	**X ← X * C**
ADD	X, R0	**X ← X + R0**

1-Adressmaschine

- Alle Operationen beziehen sich auf ein spezielles Register, den sogenannten **Akkumulator**
- Befehle enthalten maximal eine Operandenadresse, der zweite Operand befindet sich stets im Akkumulator
- Das Ergebnis wird in den Akkumulator gespeichert

Befehl		Kommentar (Ergebnis: X = A * B + C * C)
LOAD	A	Transferiert A in den Akkumulator AC
MULT	B	AC ⟵ AC * B
STORE	T	Transferiert AC zur Speicherstelle T
LOAD	C	Transferiert C in den Akkumulator AC
MULT	C	AC ⟵ AC * C
ADD	T	AC ⟵ AC + T
STORE	X	Transferiert Ergebnis zur Speicherstelle X

0-Adressmaschine

- Den Befehlen werden keine Adressen von Operanden übergeben (Ausnahme: POP, PUSH)
- Es gibt stattdessen **nur *eine* für jede Instruktion verwendete Adresse**: das obere Ende eines Speicherbereichs der als „Stack“ bezeichnet wird
- Bekannt als Stackmaschine (z.B. HP-3000)

Befehl		Kommentar (Ergebnis: X = A * B + C * C)
PUSH	A	Transferiert A auf den Stack
PUSH	B	Transferiert B auf den Stack
MULT		Holt A, B vom Stack und ersetzt sie durch A * B
PUSH	C	Transferiert C auf den Stack
DUP		Dupliziert das oberste Element
MULT		Holt C, C vom Stack und ersetzt sie durch C * C
ADD		Holt C*C, A*B vom Stack, ersetzt sie durch ihre Summe
POP	X	Transferiert das Ergebnis vom Stack nach X

Stackmaschinen (Stapelspeichermaschinen)

- Die meisten Befehle beziehen sich auf die oberen Einträge (meistens die oberen 2) eines Stapelspeichers
- Die oberen Einträge des Stapelspeichers (2 bis 8 oder mehr) werden oft in der CPU gehalten (wegen Zugriffsgeschwindigkeit)
- Ideal um Ausdrücke zu berechnen (Stapelspeicher hält dazwischenliegende Resultate)
  - deswegen (z.T. bis heute, oft auch im Finanzsektor) auch in bestimmten Taschenrechnern als Eingabemethode beliebt („Umgekehrte Polnische Notation“)
- Werden als effiziente Architektur in Zusammenhang mit höheren Programmiersprachen angesehen
- Auf „realer“ Hardware ist das Konzept heute weitgehend irrelevant
- Aber viele virtuelle Maschinen nutzen es: die Java Virtual Machine (JVM), CLE (Laufzeitumgebung für Microsoft .NET), PostScript,…

Tradeoffs

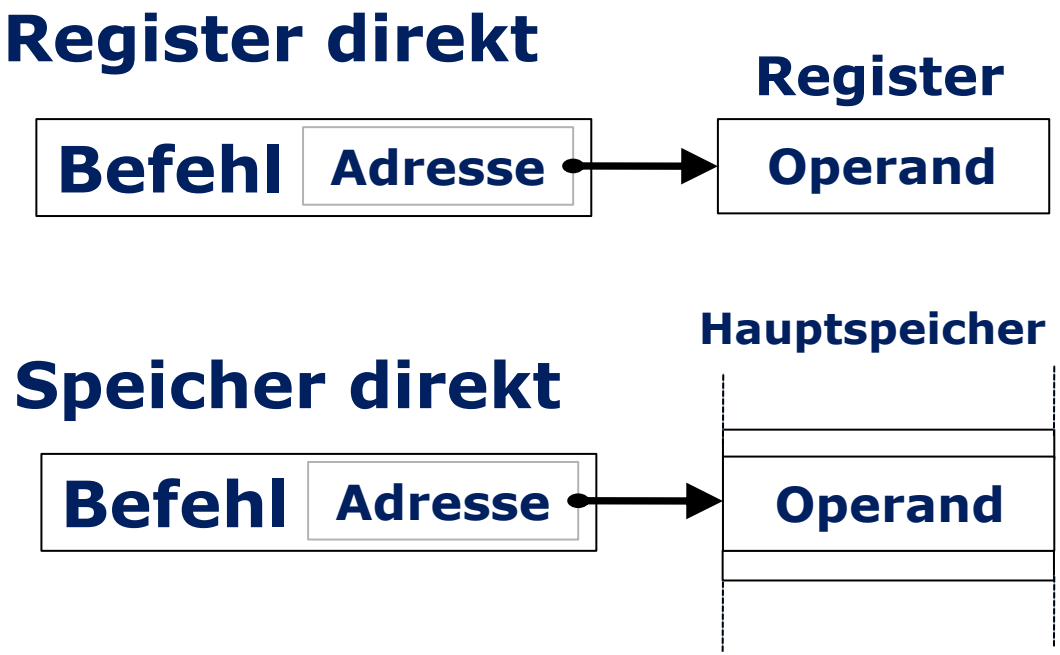
	Verschiedene n-Adressmaschinen			
Adressen	0	1	2	3
Aufteilung von 32 Bit, z.B.	32 Bit	8 Bit 24 Bit	8 12 12	8 8 8 8
Verarbeitungs-geschwindig-keit/ Befehl	OpCode/ Daten	O Adresse	O A1 A2	O A1 A2 A3
Befehle im Beispiel	8	7	5	3
Befehlsanzahl	Sehr viele	Viele	Wenige	Sehr wenige

Allgemeine Adressierungsarten

- Unmittelbare Adressierung (immediate)
  - Operand ist im Befehl enthalten

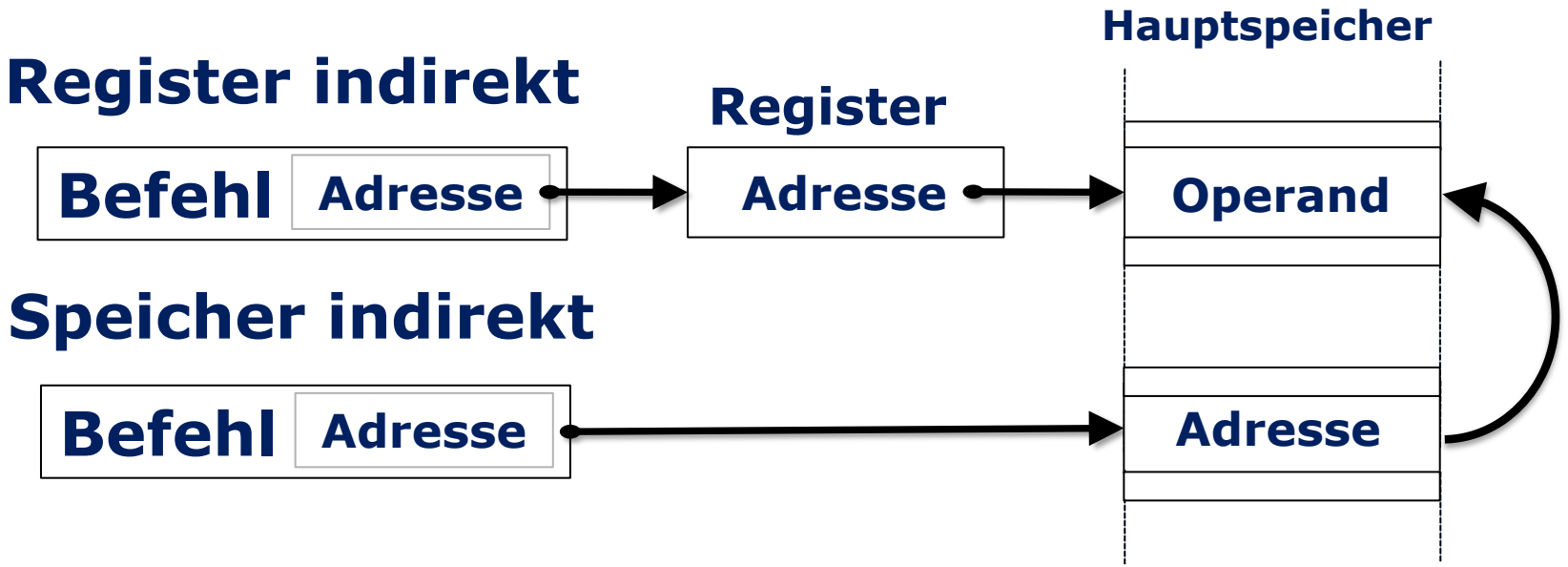
Allgemeine Adressierungsarten

- Absolute/direkte Adressierung (direct)
  - Adresse des Operanden ist Teil des Befehles
  - bei absoluten Sprüngen: Adresse des nächsten auszuführenden Befehls



Allgemeine Adressierungsarten

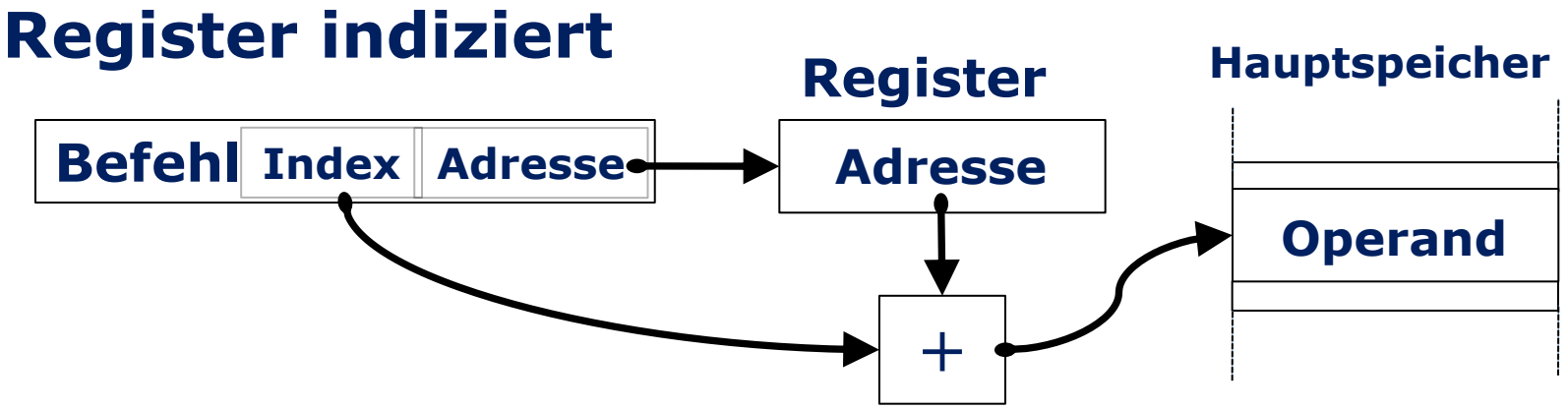
- Indirekte Adressierung (indirect)
  - der Befehl enthält die Adresse eines Registers oder einer Speicherzelle; diese(s) enthält die Adresse des Operanden



Allgemeine Adressierungsarten

- Indizierte Adressierung (indexed)
  - Effektive Adresse (EA) des Operanden wird durch Addition eines Wertes (Index X, in Befehl oder Register enthalten) und der sog. direkten Adresse (in Register/Speicherzelle enthalten) berechnet:

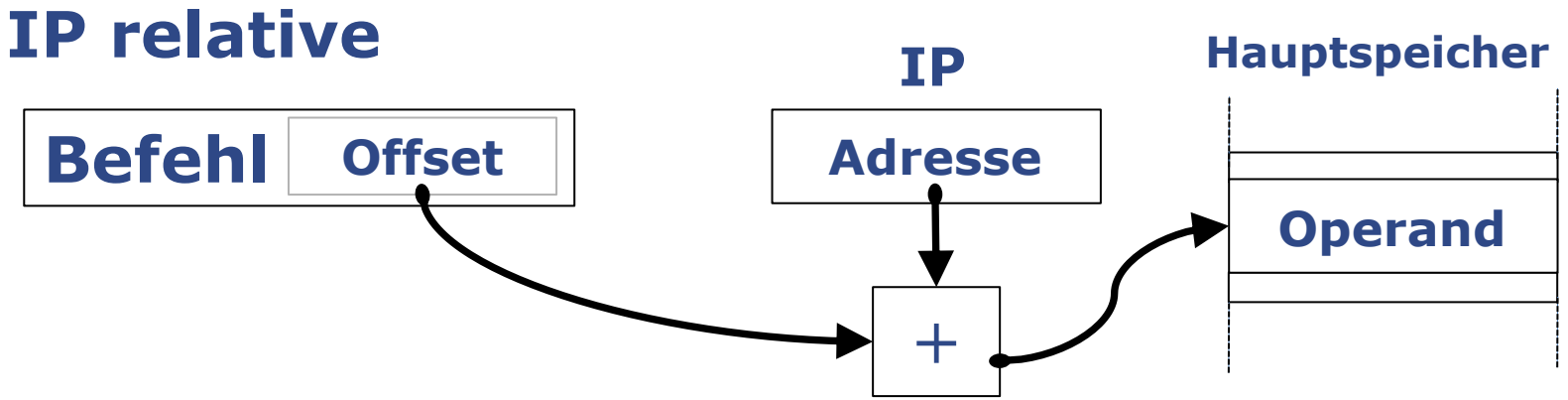
    EA = X + DA



Allgemeine Adressierungsarten

- Relative Adressierung (self-relative, relative)
  - Effektive Adresse ist die Summe des Offsets (Versatz) und des Inhalts des Befehlszzeigers (IP):

    EA = OF + IP
    - adressiert werden entweder Daten oder, bei relativen Sprüngen, Befehle



Allgemeine Adressierungsarten

- Basisadressierung (base)
  - Effektive Adresse des Operanden wird durch die Addition der direkten Adresse (im Befehl enthalten) und des Inhalts eines Basisregisters berechnet:

$$EA = B + DA$$

- Implizite Adressierung (implied)
  - Adresse des Operanden ergibt sich aus dem Befehl (z. B. wird bei einem Ein-Adress-Rechner der Akkumulator automatisch adressiert)

Allgemeine Adressierungsarten

- Segmentierte Adressierung (augmented)
  - Effektive Adresse ergibt sich aus dem Aneinanderfügen des Inhalts eines Segment-Adressregisters (SAR) und der direkten Adresse:

    EA = SAR || DA

    (z.B. SAR spezifiziert dabei einen Speicherbereich (Speicherseite) und DA ist eine Adresse innerhalb dieser Seite)

Allgemeine Adressierungsarten

- Block Adressierung
  - Adresse des ersten Wortes im Block ist gegeben
  - Anzahl der Wörter wird bestimmt durch
    - den Befehl
    - Angabe der letzten Adresse
    - ein besonderes end-of-block Zeichen
    - eine feste Länge

Architekturtypen

- Stack-Architekturen
- Register-Register-Architekturen
- **Register-Speicher-Architekturen** (moderne Prozessoren)
- Speicher-Speicher-Architekturen
- Markierte Architekturen (tagged architectures)
  - z.B. Lisp-Maschine, bei der die Speicherzellen Daten und Beschreibungen der Daten (Typen, Objektzugehörigkeit, Behandlungsvorschriften, ...) enthalten
- Architekturen sind unterscheidbar nach
  - Leistung, Effizienz
  - Design Komplexität
  - Einfachheit von Programmierung, der Parameterübergabe und der Unterprogrammaufrufe
  - Rekursionsmöglichkeiten

General Purpose Register Maschinen (GPRM)

- Bei General Purpose Register Maschinen ist der CPU-interne Speicher als ein Satz von Registern organisiert, die meist für alle Befehle gleichermaßen verfügbar sind
- Wiederholt benutzte Operanden werden durch das Programm bevorzugt in Register platziert
- Befehlsgröße wird reduziert
- Anzahl der Hauptspeicherzugriffe wird reduziert
- Alle modernen Universal-Mikroprozessoren

Zusammenfassung

- Adressierungsarten haben großen Einfluss auf Architektur, Leistung, Effizienz, Adressierbarkeit und Kosten des Computers
- Deswegen gibt es eine Vielfalt an Adressierungsarten
- Zahl der Adressen in einer (ALU-)Instruktion (n-Adressmaschine) bestimmt Programmierbarkeit, Adressierbarkeit, Leistung, Komplexität und Flexibilität des Computers
- Die Wahl der Adressierungsarten ist ein Kompromiss zwischen Programmierbarkeit, Hardwarekomplexität, Leistungsfähigkeit und Adressierbarkeit