

Kapitel 4

Befehlssatzarchitektur *(Instruction Set Architektur ISA)* **Die Hardware-Software-Schnittstelle**

- Datentypen, Datenformate
- Befehlsformat, Befehlssatz
- Adressierungsarten
- Diskussion: RISC & CISC; Fallstudien (MIPS)



Datentypen

□ Fallstudie: IA-32

- Ordinal bzw. unsigned integers (vorzeichenlose Dualzahl) in den Standardformaten;
- Integer (2-Komplementzahl) in den Standardformaten;
- Packed BCDs (binär codierte Dezimalzahl in gepackter Darstellung);
- Unpacked BCDs (binär codierte Dezimalzahl in ungepackter Darstellung);
- Near Pointer (effektive Adresse innerhalb eines Segments);
- Far Pointer (logische Adresse, die sich aus dem 16-Bit breiten Segmentselektor und dem 32-Bit breitem Offset zusammensetzt);
- Bit Fields (Bitfelder)
- Strings (Folge von Bits, Bytes, Wörtern oder Doppelwörtern, wobei ein Bit-String an jeder Position in einem Byte beginnen kann und bis zu $2^{32}-1$ Bits enthalten kann und Byte-String Bytes, Words oder Doublewords enthalten kann in einem Bereich von 0 bis $2^{32}-1$ Bytes)
- Floating-Point Data Types (Gleitkomma-, Integer und BCD-Zahlen)



Datentypen

□ Fallstudie: MIPS64 Architektur

- MIPS64 Operationen arbeiten auf 64 Bit Integer und 32 oder 64 Bit Gleitkommadaten
- Byte ,Half Word und Word Daten werden in GPRS geladen mit Null oder Vorzeichenerweiterung



Speicheradressierung

□ Datenzugriff

➤ Byteadressierbarer Speicher

- direkt zugreifbar im Speicher ist das Byte, Halbwort oder das Wort, wobei sich die Adressen, unabhängig vom Datenformat auf Bytegrenzen beziehen (Byteadressen);

➤ Wortorganisierter Speicher

- die Zugriffsbreite ist (bei optimaler Auslegung) gleich der Datenbusbreite (z.B. 32 Bit bei 32-Bit Mikroprozessoren oder 64 Bit bei 64/32- bzw. 64-Bit Prozessoren);
- die Zugriffsbreite kann bei einem Prozessor, der eine dynamische Anpassung der Datenbusbreite vorsieht, auf 16 Bit (halbwortorganisiert) oder 8 Bit (byteorganisiert) reduziert sein;



Speicheradressierung

□ Datenzugriff

➤ Ausrichten der Daten im Speicher (data alignment)

- ein Datum in einem Format bestehend aus s Bytes ist im Speicher ausgerichtet abgelegt, wenn seine Adresse A ein ganzzahliges Vielfaches von s ist, d. h. wenn $A \bmod s = 0$ ist;
- ein Speicherzugriff pro Operand: Gilt nur dann, wenn die Operandenbreite \leq Wortbreite des Speichers ist.



Speicheradressierung

□ Datenzugriff

- Nicht ausgerichtete Daten (data misalignment)
 - Speicherung von Operanden in den Datenformaten Halbwort, Wort, Doppelwort etc. an beliebigen Byteadressen;
 - nicht bei allen Prozessoren erlaubt;
 - Vorteil:
 - lückenlose Nutzung des Speichers bei beliebiger Mischung der Datenformate;
 - Nachteil:
 - zusätzlich Speicherzugriffe
 - Beispiel Intel Pentium Pro:
 - erlaubt den Zugriff auf Operanden im Word-, Doubleword oder Quadword-Format, die über die 4-Bytes- oder 8-Bytes-Grenzen abgelegt sind, was allerdings einen zusätzlichen Taktzyklus erfordert; ein Wort, das an einer ungeraden Adresse beginnt und nicht über eine 4-Bytes-Grenze geht, wird als ausgerichtet betrachtet;
 - Beispiel MIPS R10000:
 - schreibt das Ausrichten der Daten auf vor;

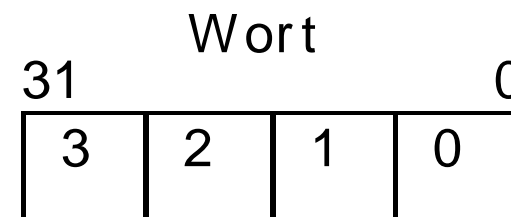
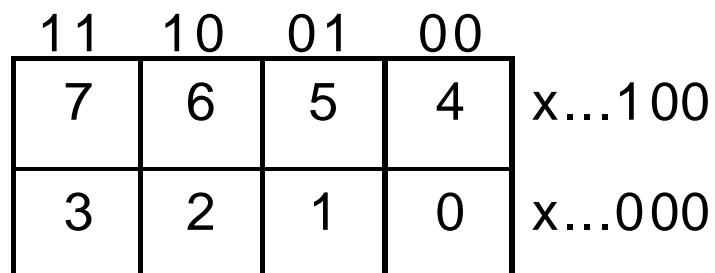


Speicheradressierung

□ Anordnung der Daten im Speicher

➤ Little Endian Ordering

- Daten in Formaten, die größer als ein Byte sind, werden so im Speicher abgelegt, dass das niedrigstwertige Byte an der niedrigstwertigen Adresse und das höchstwertige Byte an der höchstwertigen Adresse steht

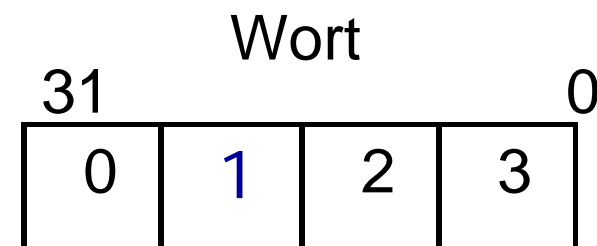
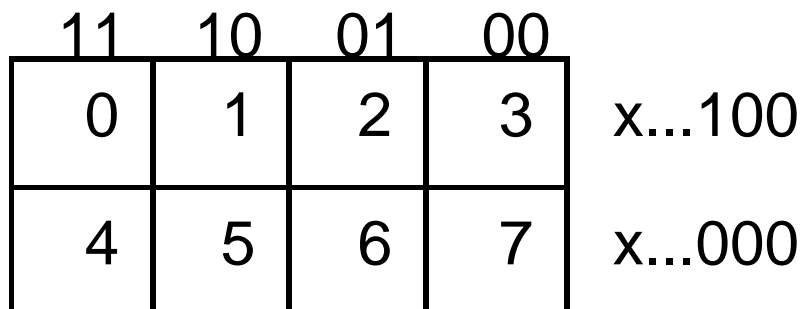


Speicheradressierung

□ Anordnung der Daten im Speicher

➤ Big Endian Byte Ordering

- Daten in Formaten, die größer als ein Byte sind, werden so im Speicher abgelegt, dass das niedrigstwertige Byte an der höchstwertigen Adresse und das höchstwertige Byte an der niedrigstwertigen Adresse steht;



Speicheradressierung

Woher der Name:

Jonathan Swift: Gulliver's Reisen (1726):

„Gulliver gelangt in das Land Liliput, dessen Einwohner nur 6 Zoll groß sind. Dort war es verboten, ein Ei an der stumpfen Seite zu öffnen, weil sich der Prinz des Landes dabei in den Finger geschnitten hatte. Ein Teil der Bevölkerung, die BIGENDIANS, wollte das nicht hinnehmen, es gab 6 Rebellionen, ein Kaiser starb, einer musste gehen. Die BIGENDIANS wurden aus dem öffentlichen Dienst ausgeschlossen, 100 BIGENDIAN-Bücher verboten. Viele BIGENDIANS retteten sich in das benachbarte Blefescu. Es kam zum Krieg, der 3 Jahre dauerte und Liliput 30000 Soldaten, vierzig große und noch mehr kleinere Schiffe kostete. Die Invasion von Liliput konnte nur dadurch verhindert werden, dass Gulliver („der Menschenberg“) die feindliche Flotte raubte....“



Speicheradressierung

□ Adressierungsarten

- Spezifikation der Adresse eines Objekts (Konstante, Register, Speicherzelle);
- neben der Angabe der Speicheradresse direkt im Befehlswort, sehen Prozessoren verschiedene Möglichkeiten der **Adressmodifikationen** vor;
- Berechnung der effektiven Adresse (tatsächliche Adresse) aus mehreren Teilen, die im Befehlswort, in Registern oder Speicherzellen stehen, während der Befehlsausführung (Adressrechnung zur Laufzeit, *dynamische Adressrechnung*)



Architektur (ISA)

□ Programm-, Prozess-, Maschinenadresse

➤ Programmadresse:

- In Befehlen und als Adresswerte im Programm vorliegende Adressen
- Nach Vorgaben des Programms erzeugt der Prozessor aus Programmadressen Prozessadressen
 - Indexmodifikation
 - Substitution,
 - (Befehlszähler) relativer Adressierung
 - „offener“ Basisadressierung



Architektur (ISA)

□ Programm-, Prozess-, Maschinenadresse

➤ Prozessadresse (effektive Adresse):

- Verwendet der Prozessor
- Nach Vorgaben des Betriebssystems erzeugt der Prozessor aus Prozessadressen Maschinenadressen
 - „verdeckte“ Basisadressierung
 - Seitenadressierung
- Hauptziel:
 - Beliebige Lage des Programms und seiner Werte
 - partielle Lagerung im Speicher



Architektur (ISA)

□ Programm-, Prozess-, Maschinenadresse

➤ Maschinenadresse

- Verwendet der Prozessor gegenüber Hauptspeicher



4.2 Befehlssatz

Welche Operationen können auf den Daten ausgeführt werden?

- Der **Befehlssatz** (*instruction set*) legt die Grundoperationen eines Prozessors fest.
- **Befehlsarten:**
 - Transportbefehle
 - Arithmetisch-logische Befehle
 - Schiebe- und Rotationsbefehle
 - Multimediabefehle
 - Gleitkommabefehle
 - Programmsteuerbefehle
 - Systemsteuerbefehle
 - Synchronisationsbefehle



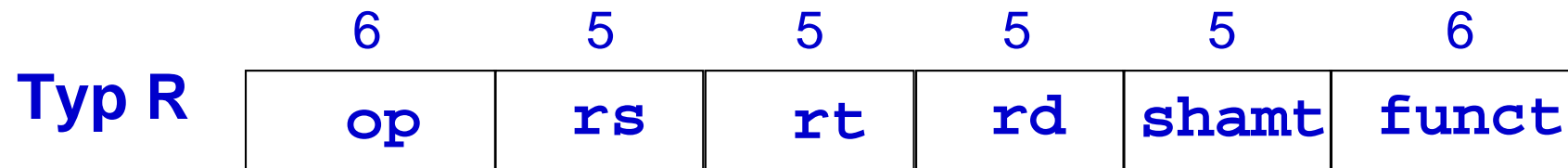
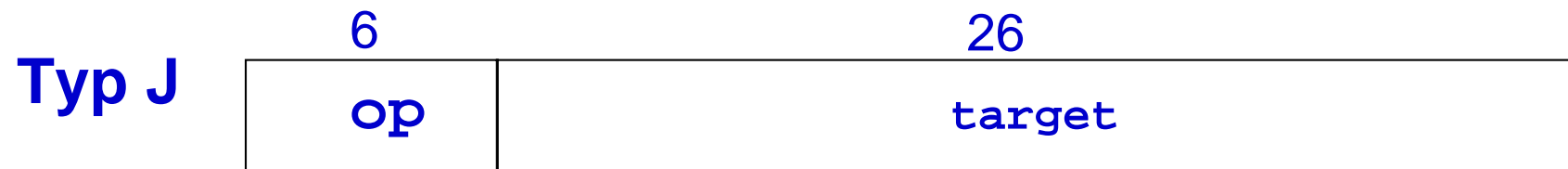
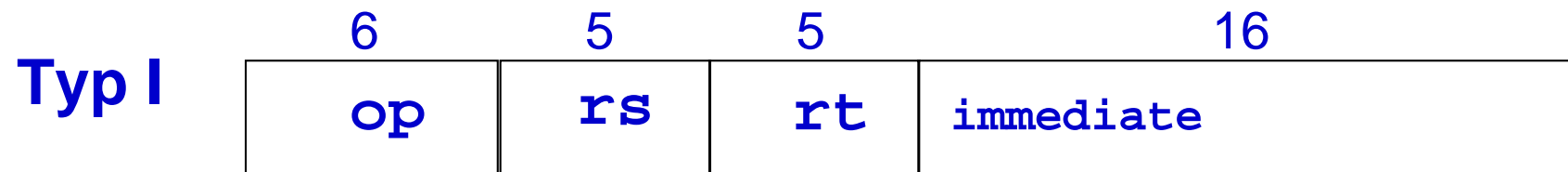
4.2 Befehlsformate

- ❑ Das **Befehlsformat** (*instruction format*) definiert, wie die Befehle codiert sind.
- ❑ Eine **Befehlscodierung** beginnt mit dem **Opcode**, der den Befehl selbst festlegt.
- ❑ In Abhängigkeit vom Opcode werden weitere Felder im Befehlsformat benötigt.
- ❑ Art der **Adressformate** definiert vier Klassen von **Befehlssätzen**:
 - **Dreiadressformat**: Opcode Dest Src1 Src2
 - **Zweiadressformat**: Opcode Dest/Src1 Src2
 - **Einadressformat**: Opcode Src
 - **Nulladressformat**: Opcode



Befehlsformate

Der MIPS-Prozessor hat ausschließlich Befehle fester Länge (32-Bit). Die Befehle werden in Typ I, J und R unterteilt:



Befehlsformate des MIPS-Prozessors

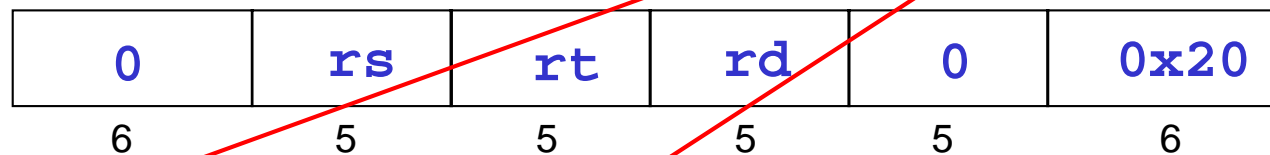
Abk.	Bedeutung
I	Immediate (direkt)
J	Jump (Sprung)
R	Register
op	6 Bit OpCode des Befehls
rs	5 Bit Kodierung eines Quellenregisters
rs	5 Bit Kodierung eines Quellenregisters oder Zielregisters
immediate	16 Bit unmittelbarer Wert oder Adressverschiebung
target	26 Bit Sprungadresse
rd	5 Bit Kodierung des Zielregisters
shamt	5 Bit Kodierung der Größe einer Verschiebung (<i>shift amount</i>)
funct	6 Bit Kodierung der Funktion (<i>function</i>)



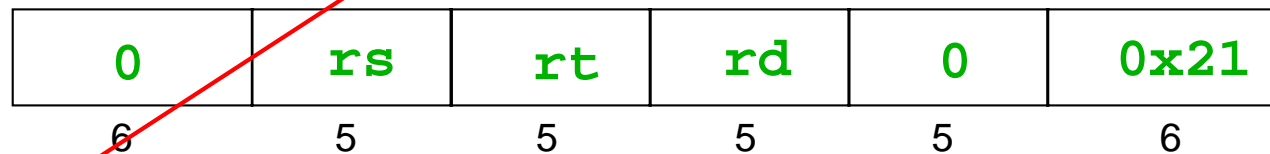
Beispiel: Additionsbefehle in MIPS

Befehlsformate (z. B. bei der Addition):

add rd,rs,rt



addu rd,rs,rt



addi rt,rs,imm



4.3 Adressierungsarten

- **Adressierungsarten:** die verschiedenen Möglichkeiten eines Prozessors die Adresse eines Operanden oder eines Sprungziels im Speicher zu berechnen.
- **Früher:** Adresse der Operanden und Sprungziele absolut im Befehl vorgegeben
 - **Nachteile:**
 - absolute Adressen müssen bereits zur Programmierzeit festgelegt werden → Programme sind lageabhängig im Speicher
 - Bei Tabellenzugriffen im Speicher muss die Adresse im Befehl geändert werden → keine Festwertspeicher als Programmspeicher möglich

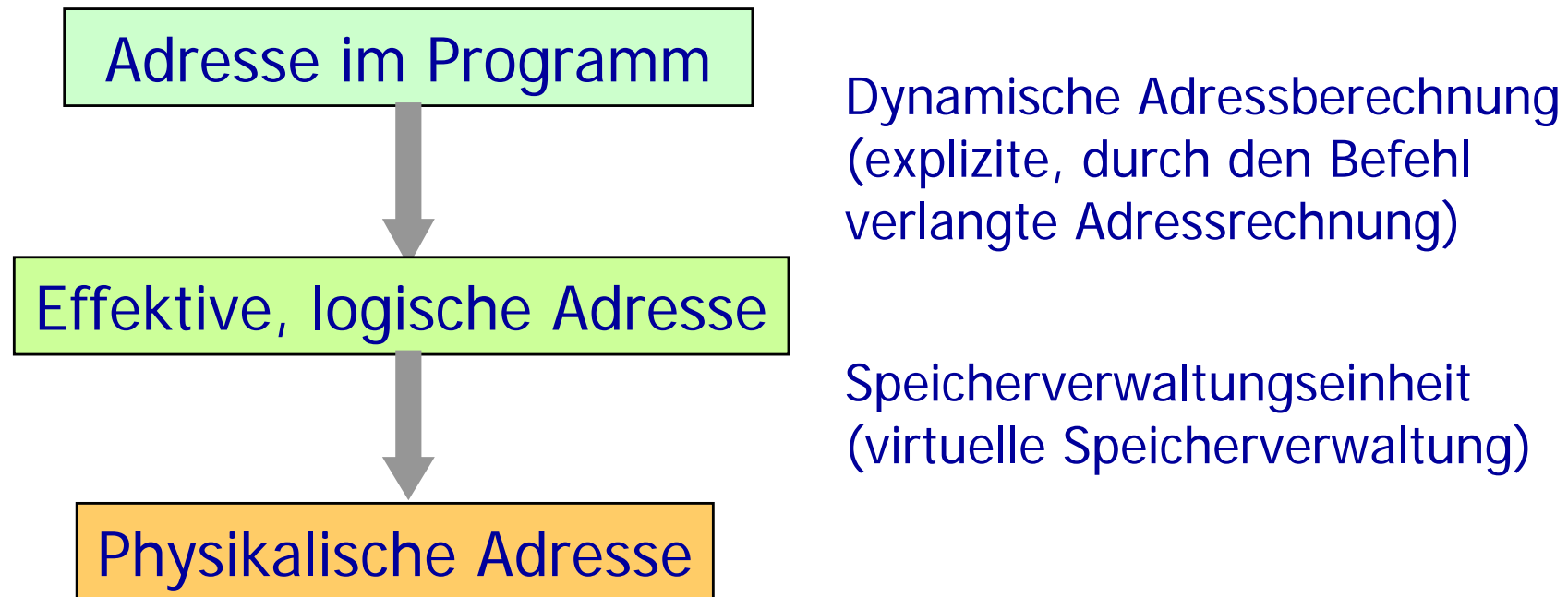


4.3 Adressierungsarten

□ Abhilfe:

- Adresse wird zur Laufzeit berechnet (dynamische Adressberechnung)

Ablauf der Adressberechnung:



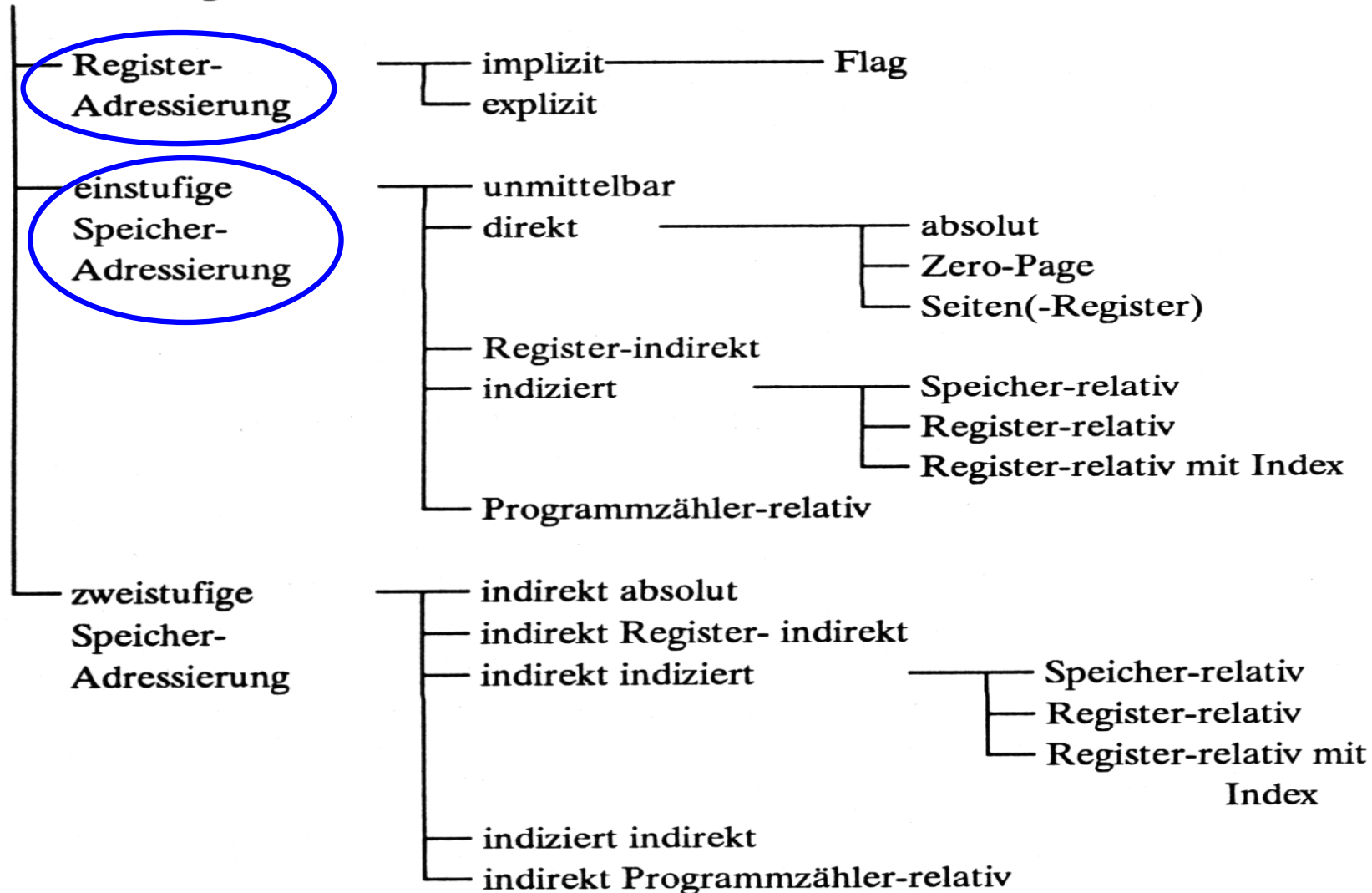
4.3 Adressierungsarten

- ❑ **Effektive Adresse:** die durch die Adressierungsart spezifizierte Speicheradresse im Hauptspeicher
 - Eine effektive Adresse entsteht im Prozessor nach Ausführung der Adressrechnung.
- ❑ Bei virtueller Speicherverwaltung gilt:
effektive Adresse = logische Adresse,
diese wird weiteren Speicherverwaltungsoperationen in einer **Speicherverwaltungseinheit** (*Memory Management Unit MMU*) unterworfen, um letztendlich eine **physikalische Adresse** zu erzeugen, mit der dann auf den Hauptspeicher zugegriffen wird.
- ❑ Im folgenden betrachten wir nur die Erzeugung einer effektiven Adresse aus den Angaben in einem Maschinenbefehl.



4.3 Adressierungsarten

Adressierungsarten



4.3.1 Register Adressierung

4.3.1 Register-Adressierung

Operand steht bereits im Register

→ kein Speicherzugriff erforderlich

4.3.2 Einstufige Speicher-Adressierung

Eine Adressberechnung zur Ermittlung der effektiven Adresse notwendig, d. h. keine mehrfachen Speicherzugriffe zur Adressermittlung

4.3.3 Zweistufige Speicher-Adressierung

Mehrere sequentielle Adressberechnungen und Speicherzugriffe. Ergebnis der ersten Berechnung liefert die Adresse einer Speicherzelle, deren Inhalt wieder eine Adresse oder ein Offset zur weiteren Berechnung ist



4.3.1 Register Adressierung

4.3.1 Register-Adressierung

Operand steht bereits im Register →
kein Speicherzugriff erforderlich

- ❑ Implizite Adressierung
- ❑ Flag-Adressierung
- ❑ Explizite Register-Adressierung



Implizite Adressierung

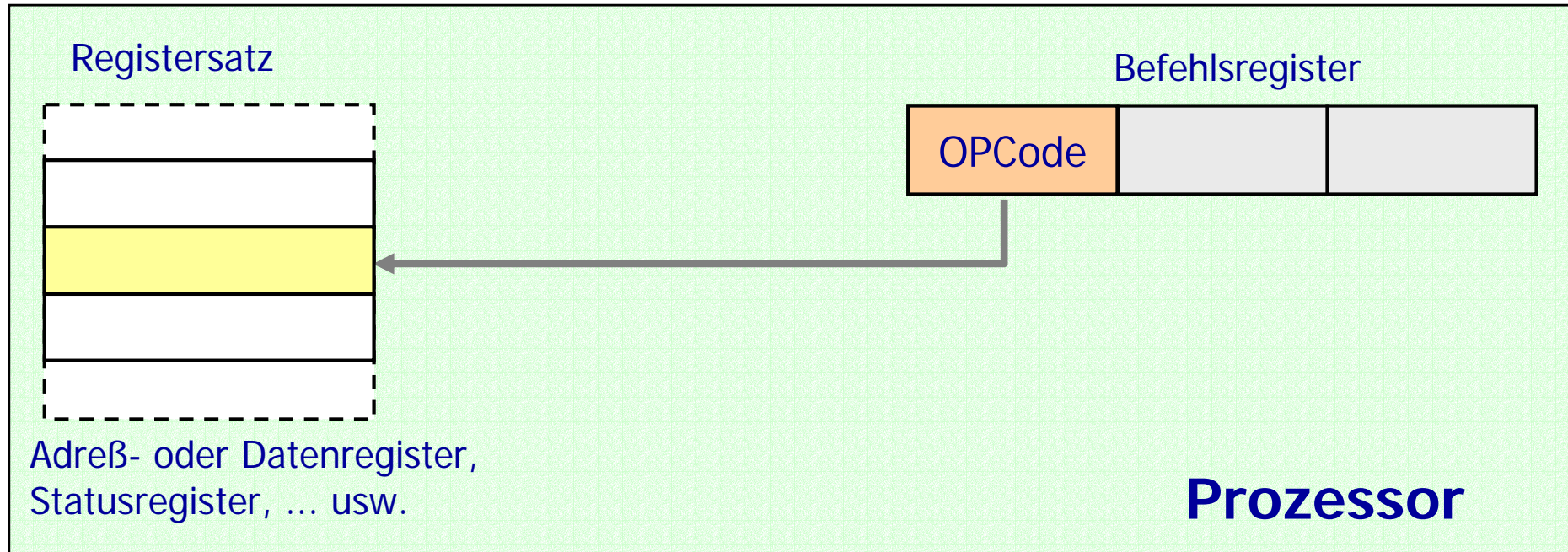
□ Implizite Adressierung (*inherent Adressierung, implied-, inherent addressing*)

Die Nummer, d. h. die effektive Adresse des angesprochenen Registers ist **codiert im Operations-Feld des OpCodes** enthalten

- **Assemblerschreibweise:**
 <Mnemonic> A (A Akkumulator)
- **Effektive Adresse:**
 EA ist codiert im OpCode enthalten



Implizite Adressierung



Beispiel:

LSRA (*logical shift right accumulator*)

(Verschiebe den Inhalt des Akkumulators A eine Bitposition nach rechts)



Flag-Adressierung

□ **Flag-Adressierung:**

Sie ist ein Spezialfall der impliziten Adressierung. Bei ihr wird nicht ein ganzes Register angesprochen, sondern nur ein einzelnes Bit (Flag) in einem Register

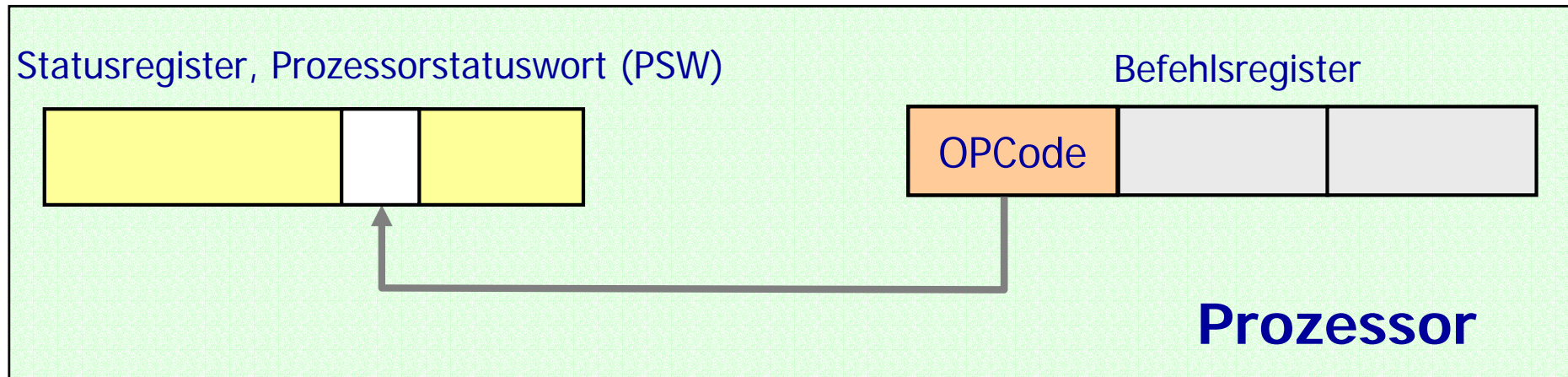
➤ **Assemblerschreibweise:**

- SE <flag> (Flag setzen)
- CL <flag> (Flag rücksetzen)

➤ **Effektive Adresse:** EA ist codiert im OpCode enthalten



Flag Adressierung



Beispiele:

- SEI/CLI *(set / clear interrupt flag)*
 - SEC/CLC *(set / clear carry flag)*
- (Setzen / Zurücksetzen des Interrupt Enable Flags bzw. des Carry Flags.)*



Explizite Register Adressierung

- **Explizite Register-Adressierung** (*register operand addressing*):

Die Adresse (Nummer) des Registers wird im Operandenfeld des Befehls angegeben

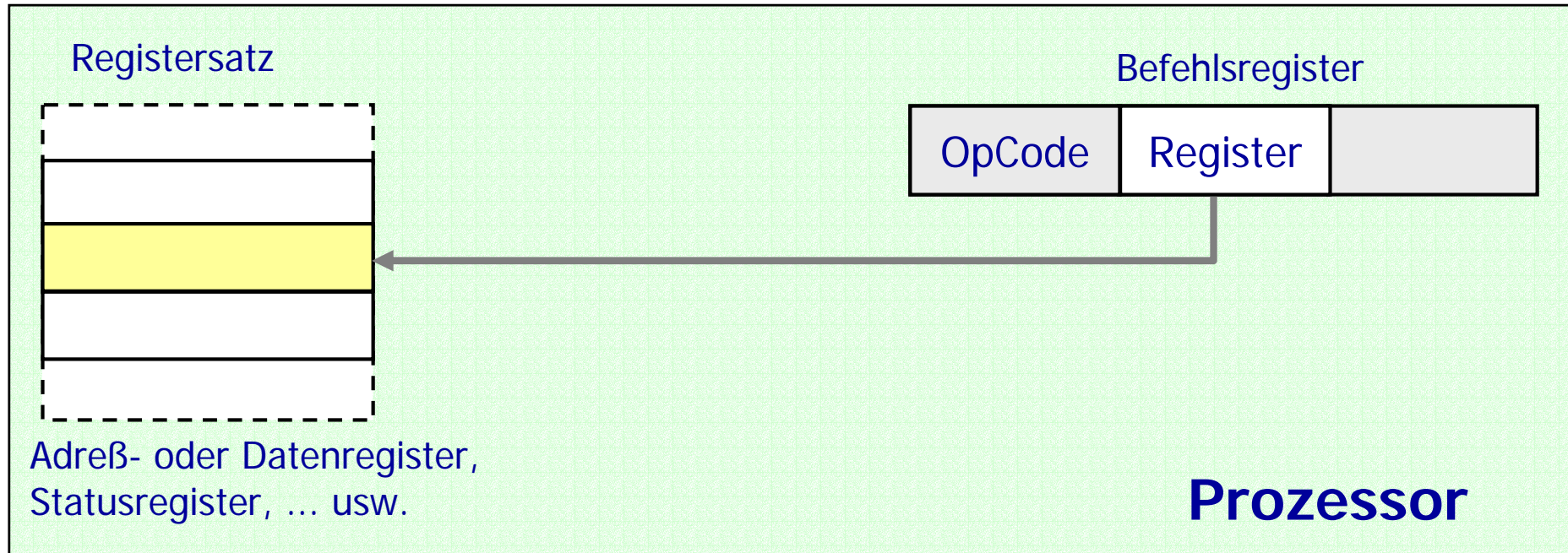
- **Assemblerschreibweise:** <Mnemo> Ri (Register i)
- **Effektive Adresse:** EA = i

Beispiel:

DEC R0 (*Decrement R0*)
(*Dekrementiere den Inhalt des Registers R0*)



Explizite Register Adressierung



- **Assemblerschreibweise:** $\langle \text{Mnemo} \rangle \text{ Ri} \quad (\text{Register } i)$
- **Effektive Adresse:** $\text{EA} = i$
- **Beispiel:** DEC R0

4.3.2 Einstufige Speicher-Adressierung

4.3.1 Register-Adressierung

Operand steht bereits im Register

→ kein Speicherzugriff erforderlich

4.3.2 Einstufige Speicher-Adressierung

Eine Adressberechnung zur Ermittlung der effektiven Adresse notwendig, d. h. keine mehrfachen Speicherzugriffe zur Adressermittlung

4.3.3 Zweistufige Speicher-Adressierung

Mehrere sequentielle Adressberechnungen und Speicherzugriffe. Ergebnis der ersten Berechnung liefert die Adresse einer Speicherzelle, deren Inhalt wieder eine Adresse oder ein Offset zur weiteren Berechnung ist



4.3.2 Einstufige Speicher-Adressierung

4.3.2 Einstufige Speicher-Adressierung

Eine Adressberechnung zur Ermittlung der effektiven Adresse notwendig, d. h. keine mehrfachen Speicherzugriffe zur Adressermittlung

- ❑ Unmittelbare Adressierung (immediate addressing)
- ❑ Direkte Adressierung (direct addressing)
 - Absolute Adressierung
 - Seiten-Adressierung
- ❑ Register-indirekte Adressierung (register indirect addressing)
- ❑ Indizierte Adressierung (indexed addressing)
 - Speicher-relative Adressierung (memory relative addressing)
 - Register-relative Adressierung (register relative addressing)
 - Register-relative Adressierung mit Index (Based indexed mode)
- ❑ Befehlszähler-relative Adressierung (PC relative addressing)

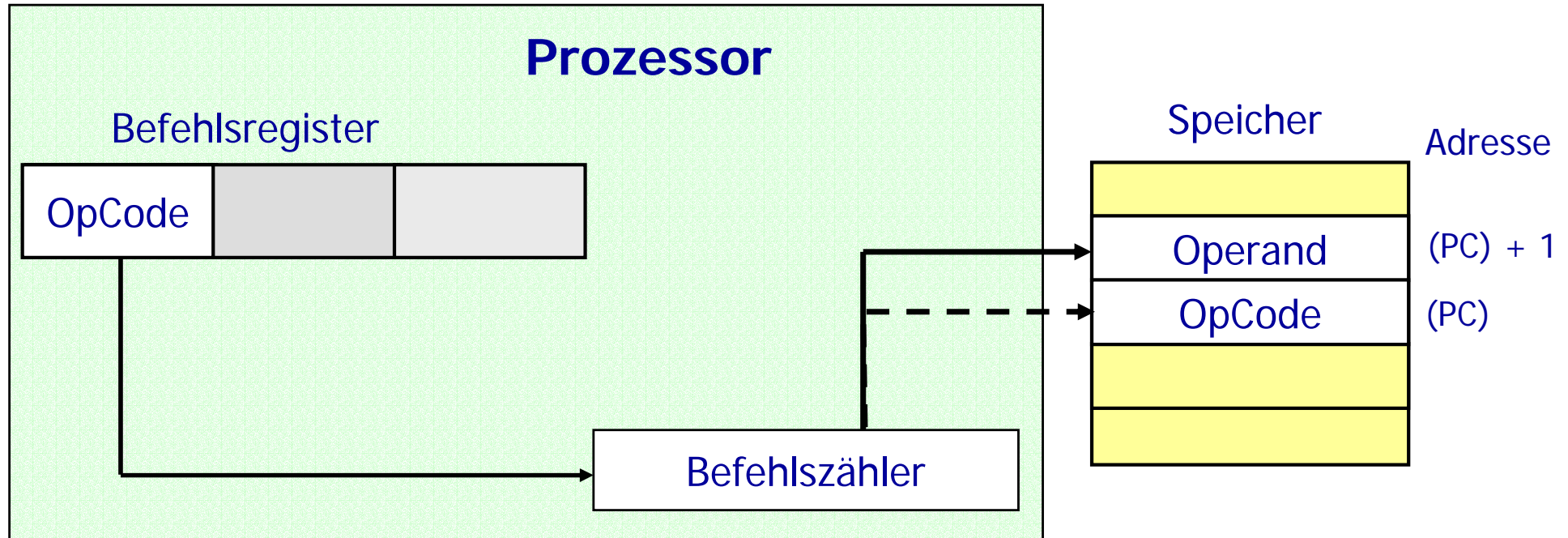


Unmittelbare Adressierung (immediate addressing)

- ❑ Der Befehl enthält nicht die Adresse des Operanden oder einen Zeiger darauf, sondern den Operanden selbst.
- ❑ OpCode und Operand belegen im Speicher hintereinander folgende Speicherworte
- ❑ **Assemblerschreibweise:** <Mnemo> #<Operand>
- ❑ **Effektive Adresse:** $EA = (PC) + 1$



Unmittelbare Adressierung (immediate addressing)



Beispiel:

LDA #A3 (load accumulator)

(Lade den Akkumulator A mit dem Hexadezimalwert A3)

