

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?

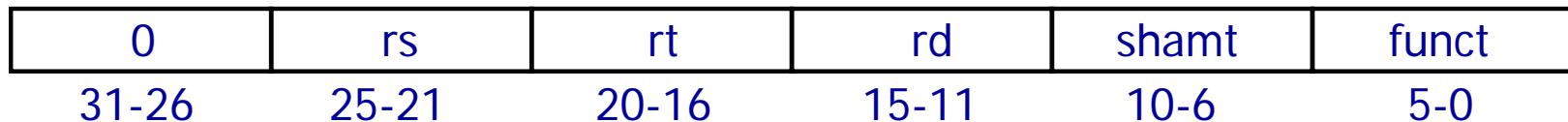
- ❑ Der **Befehlssatz** (*instruction set*) legt die Grundoperationen eines Prozessors fest.
- ❑ **Befehlsarten:**
 - Transportbefehle
 - Arithmetisch-logische Befehle
 - Schiebe- und Rotationsbefehle
 - Multimediabefehle
 - Gleitkommabefehle
 - Programmsteuerbefehle
 - Systemsteuerbefehle
 - Synchronisationsbefehle

4.2 Befehlsformate

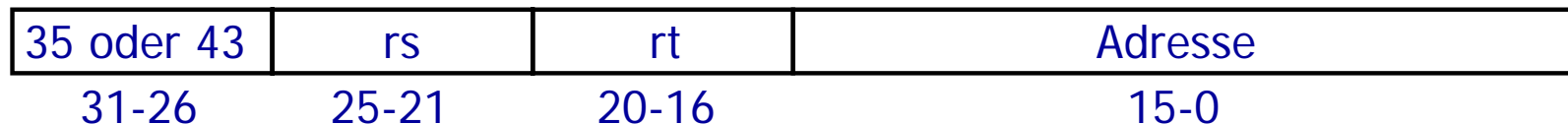
- ❑ Das **Befehlsformat** (*instruction format*) definiert, wie die Befehle codiert sind.
- ❑ Eine **Befehlscodierung** beginnt mit dem **Opcode**, der den Befehl selbst festlegt.
- ❑ In Abhängigkeit vom Opcode werden weitere Felder im Befehlsformat benötigt.
- ❑ Art der **Adressformate** definiert vier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

Befehlsformate des MIPS-Prozessors

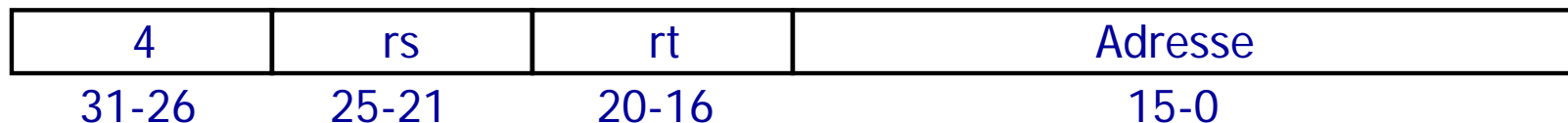
➤ Register-Register-Befehle (Typ R)



➤ Lade/Speicher Befehle (Typ I)

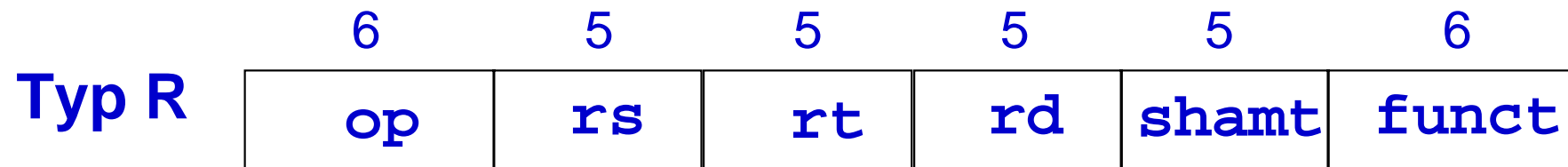
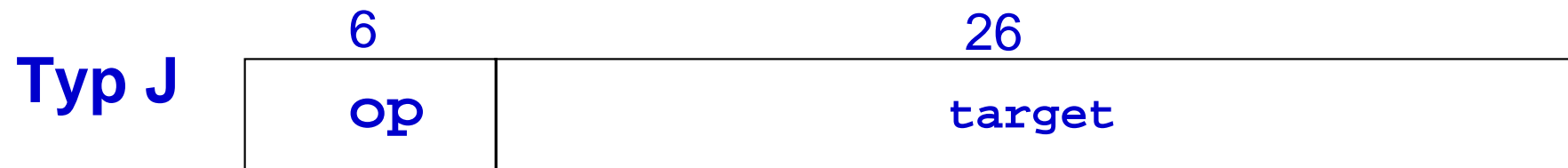
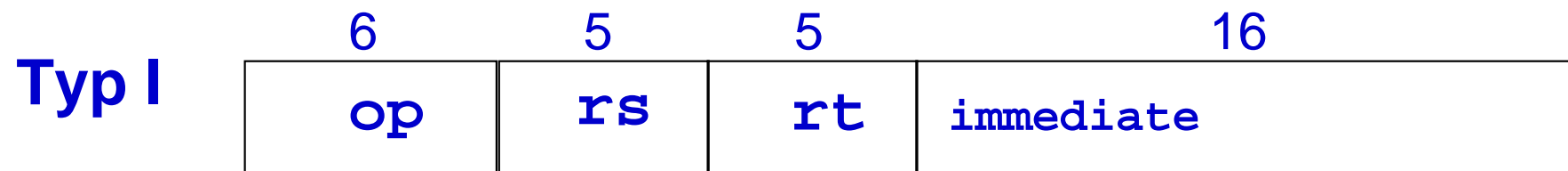


➤ Sprungbefehle (Typ J)



Befehlsformate

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



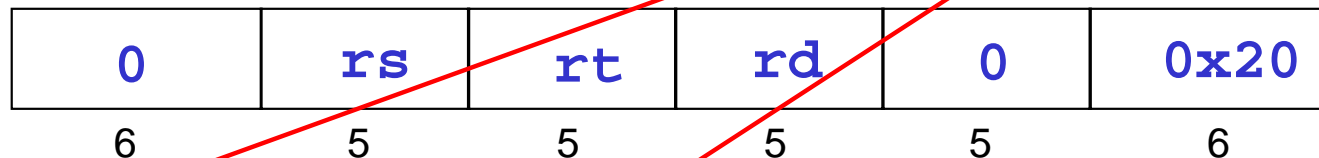
Befehlsformate des MIPS-Prozessors

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):

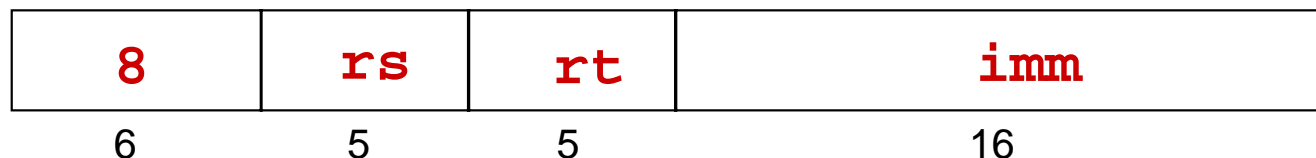
add rd,rs,rt



addu rd,rs,rt



addi rt,rs,imm



4.3 Adressierungsarten

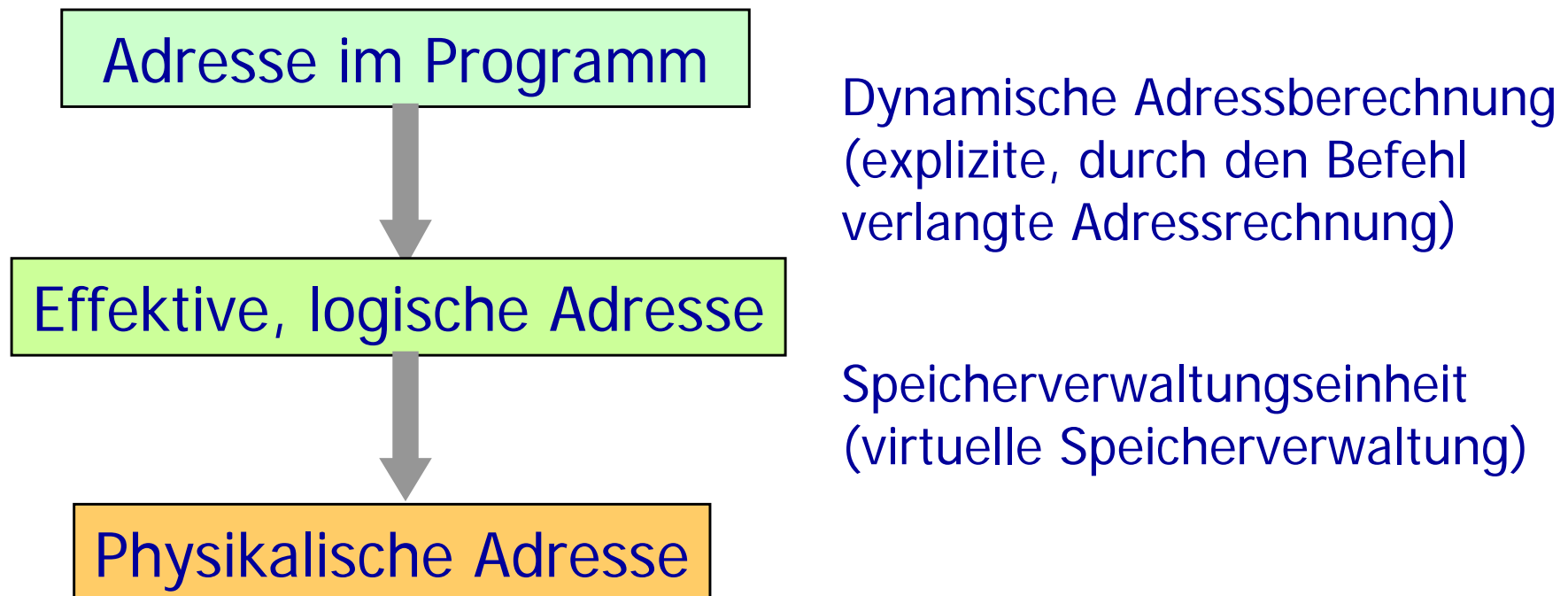
- **Adressierungsarten:** die verschiedenen Möglichkeiten eines Prozessors die Adresse eines Operanden oder eines Sprungziels im Speicher zu berechnen.
- **Früher:** Adresse der Operanden und Sprungziele absolut im Befehl vorgegeben
 - **Nachteile:**
 - absolute Adressen müssen bereits zur Programmierzeit 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:

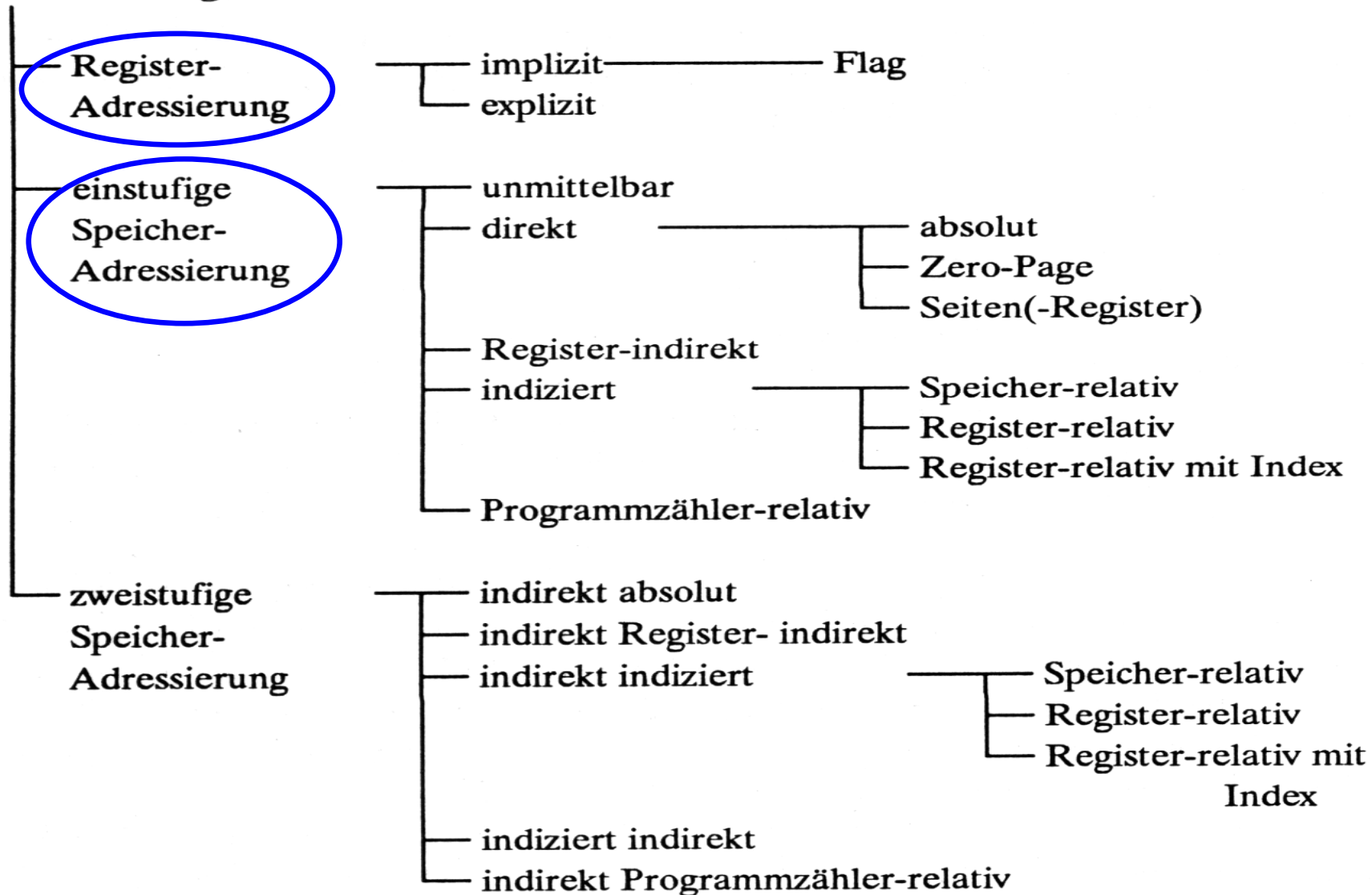


4.3 Adressierungsarten

- **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

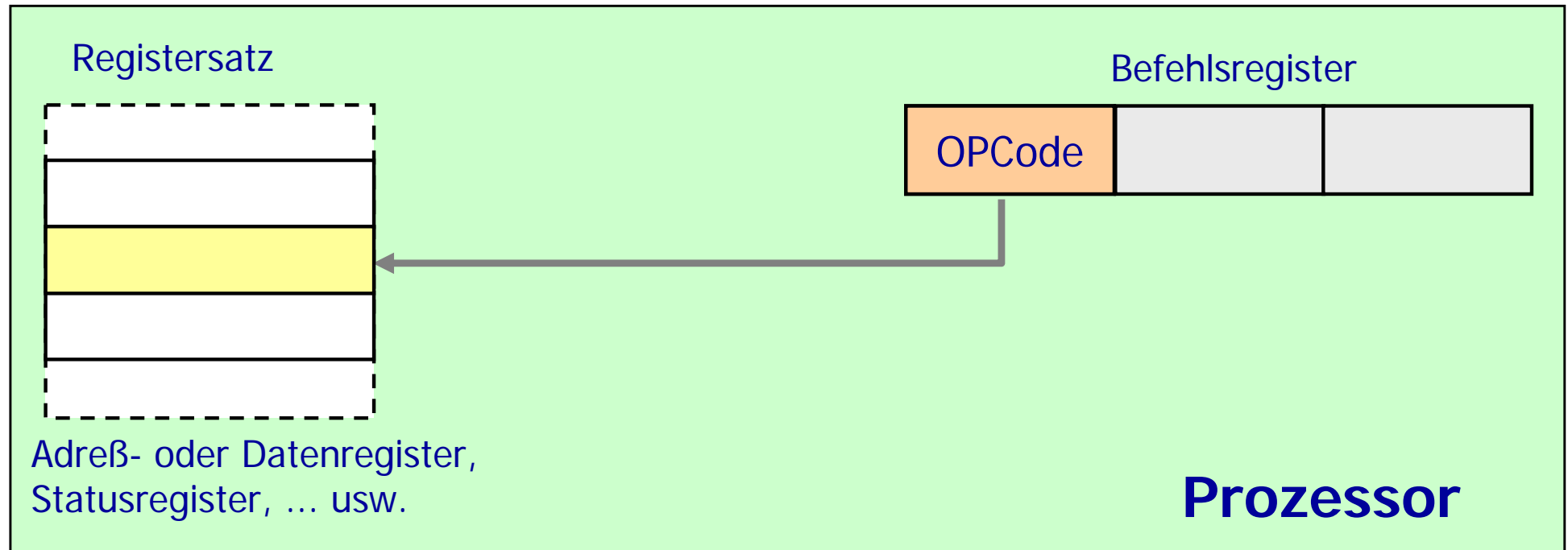
Implizite Adressierung

- **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:**
 <Mnemo> 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:

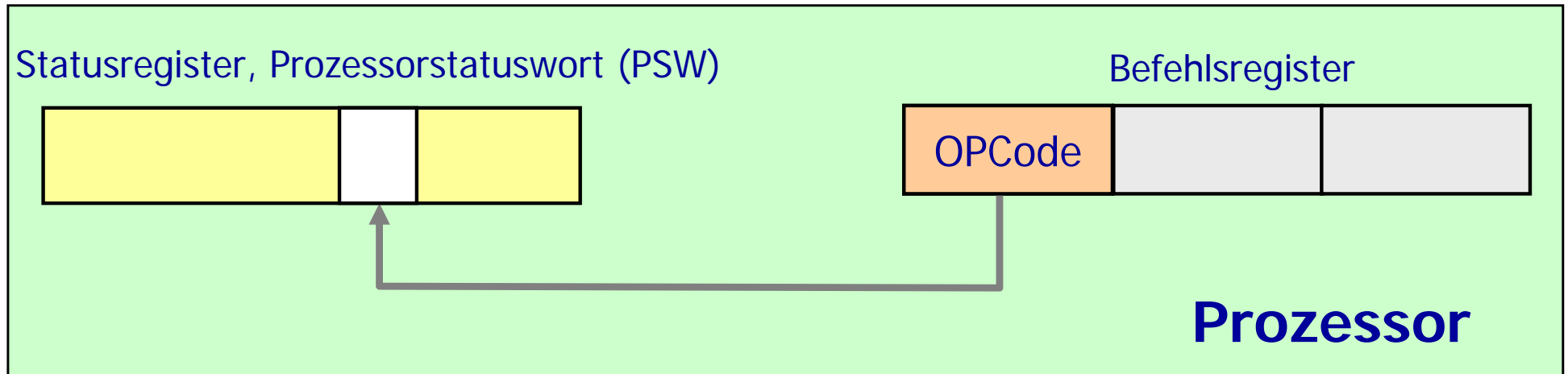
Sie ist ein Spezialfall der impliziten Adressierung. Bei ihr wird nicht ein ganzes Register angesprochen, sondern nur ein einzelnes Bit (Flag) in einem Register

➤ 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

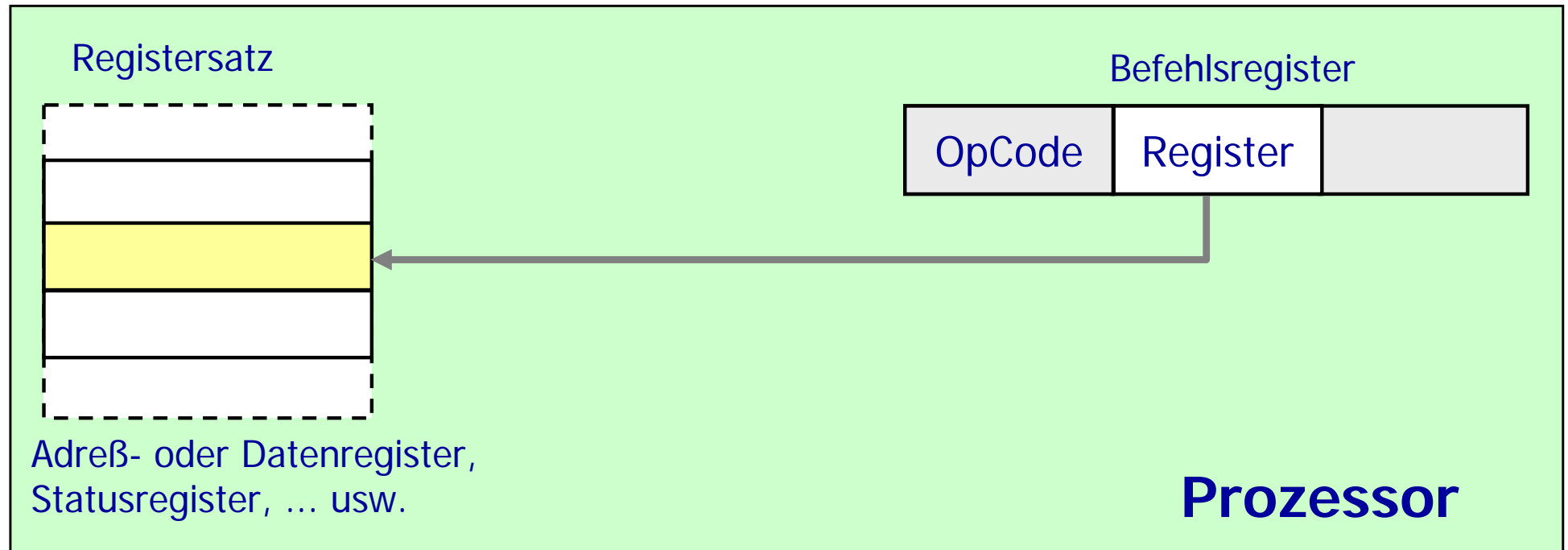
- **Assemblerschreibweise:** <Mnemo> Ri (Register i)
- **Effektive Adresse:** EA = i

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

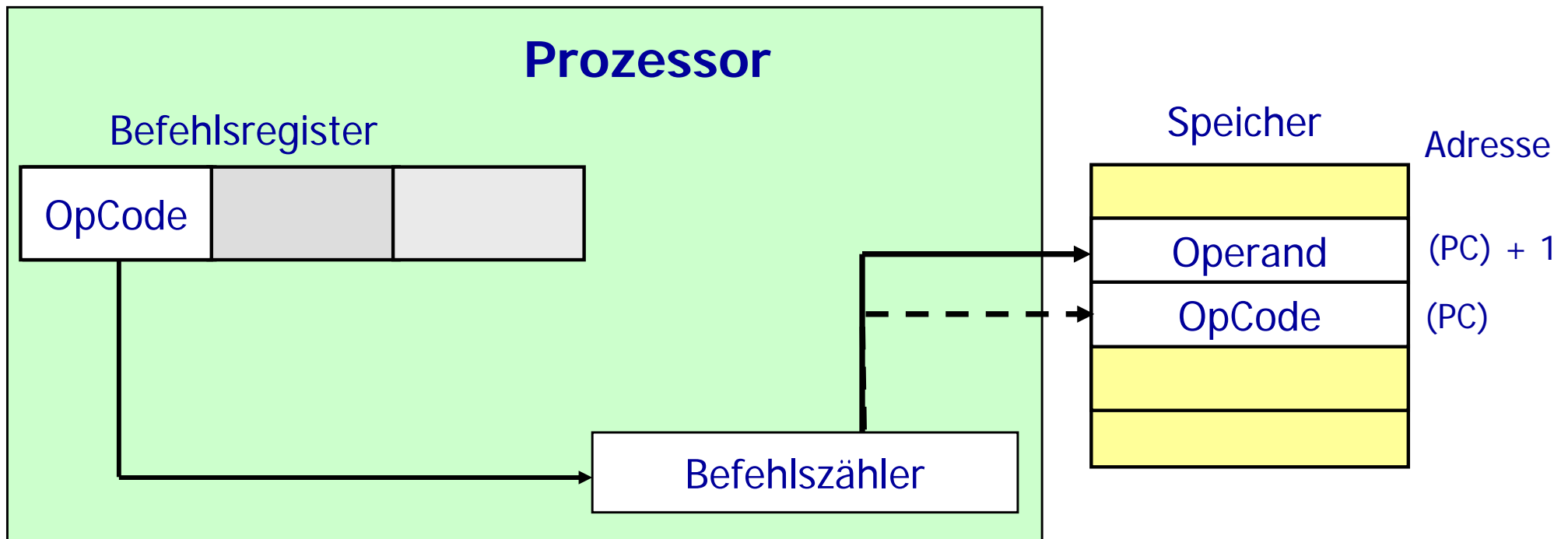
Eine Adressberechnung zur Ermittlung der effektiven Adresse notwendig, d. h. keine mehrfachen Speicherzugriffe zur Adressermittlung

- ❑ 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:** $\langle \text{Mnemo} \rangle \ \# \langle \text{Operand} \rangle$
- ❑ **Effektive Adresse:** $EA = (PC) + 1$

Unmittelbare Adressierung (immediate addressing)



Beispiel:

LDA #\$A3 (load accumulator)

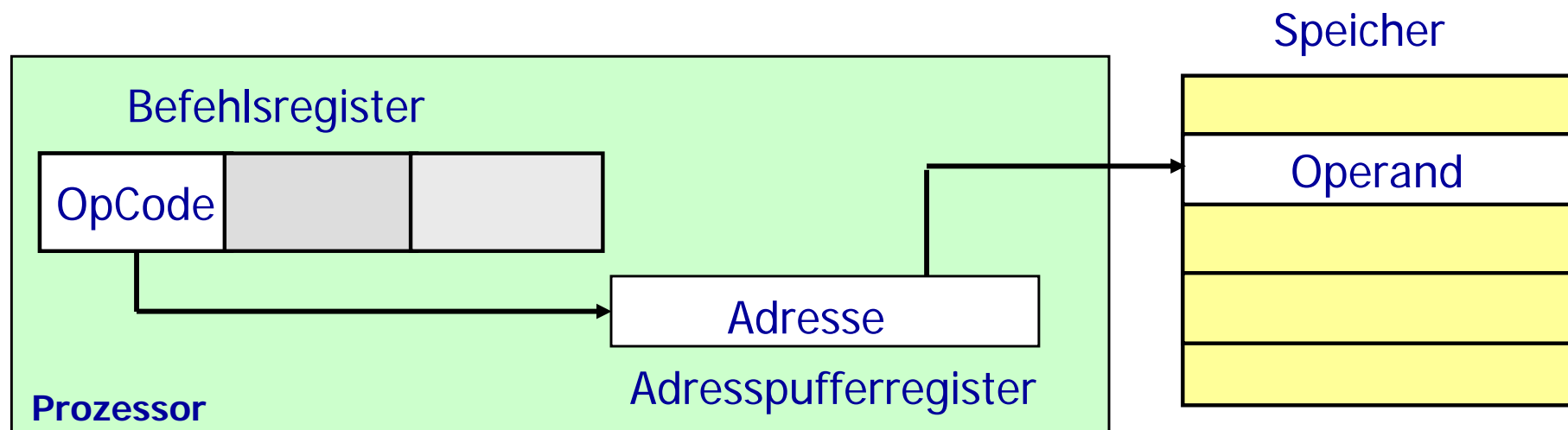
(Lade den Akkumulator A mit dem Hexadezimalwert \$A3)

Direkte Adressierung (*direct addressing*)

- ❑ Der Befehl enthält im Speicherwort nach dem OpCode die logische Adresse des Operanden, aber keine weiteren Vorschriften zu deren Manipulation, z.B. durch die Addition mit einem Registerinhalt
- ❑ **Assemblerschreibweise:** <Mnemo> <Adresse>
- ❑ **Effektive Adresse:** $EA = ((PC) + 1)$
- ❑ Zwei Fälle können unterschieden werden:
 - Absolute Adressierung
 - Seiten-Adressierung

Absolute Adressierung (*extended direct addressing*)

- Der Befehl enthält im Speicherwort, das dem OpCode folgt, die absolute, d. h. vollständige Adresse des Operanden im (logischen) Adressraum



Beispiel:

JMP \$07FE (jump)

(Springe zur Adresse \$07FE)

Seiten-Adressierung (*direct page addressing*)

- Im Befehl steht als Kurz-Adresse nur der niederwertige Teil der Operandenadresse (*Low-Adresse*)

- **Zero-page-Adressierung:**

der höherwertige Adressteil wird durch die entsprechende Anzahl von '0'-Bits ersetzt. Dadurch wird im Adressraum nur die "unterste Seite" (*zero page*) angesprochen

- **Seiten-Register-Adressierung** („*direct*“ *page register addressing*):

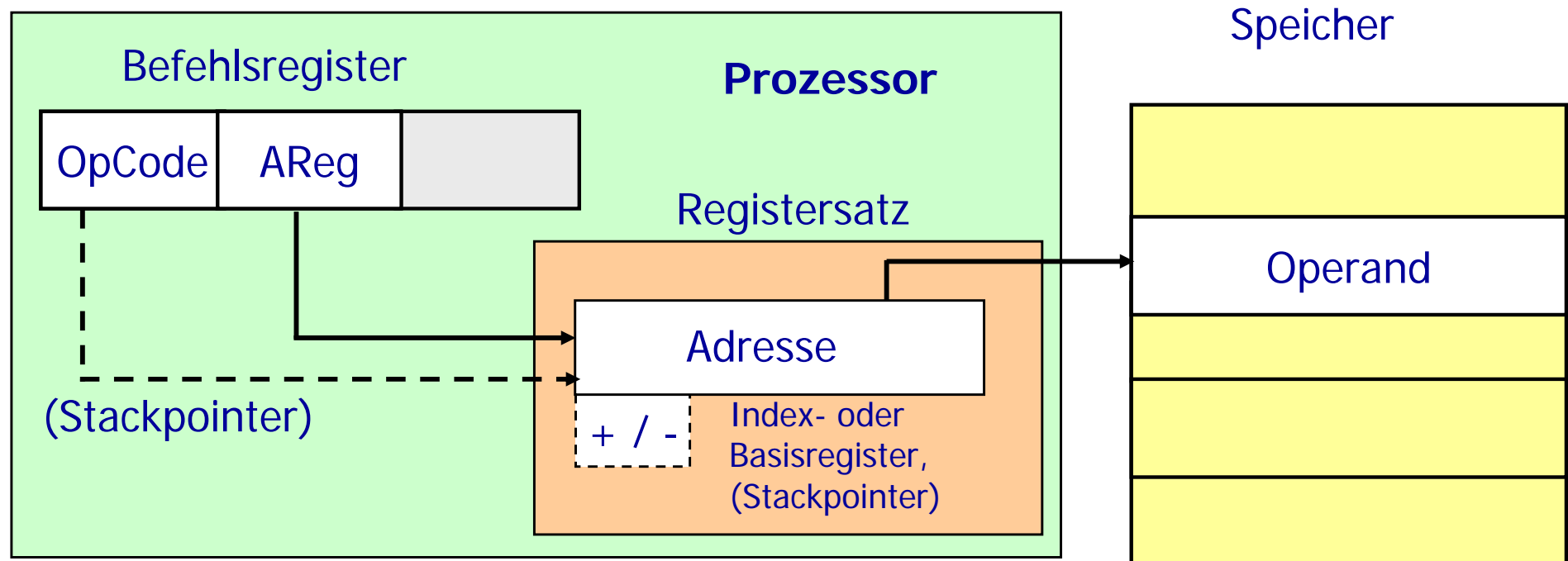
der höherwertige Adressteil wird in einem Register des Prozessors (DP-Register, direct page register) zur Verfügung gestellt

Register-indirekte Adressierung

Register-indirekte Adressierung (*register indirect addressing*)

- ❑ Hier enthält das durch seine Nummer im Register-Feld des OpCodes angegebene Adressregister die Adresse des Operanden (*pointer*, "Zeiger", deshalb auch: Zeigeradressierung)
- ❑ **Assemblerschreibweise:** <Mnemo> (Ri)
- ❑ **Effektive Adresse:** $EA = (Ri)$

Register-indirekte Adressierung



Beispiel:

LD R1, (A0) (*load*)

(Lade das Register R1 mit dem Inhalt des durch das Adressregister A0 gegebenen Speicherwortes)

Register-indirekte Adressierung

Bei der im Register stehenden Adresse handelt es sich oft um die Anfangs- oder Endadresse eines Tabellenbereichs im Speicher ➔ Registerinhalt automatisch modifizieren

- **postincrement:** Nach der Ausführung des Befehls wird der Inhalt des Registers (um 1) erhöht und zeigt danach auf die nachfolgende Speicherzelle
 - **Assemblerschreibweise:** $\langle \text{Mnemo} \rangle (\text{Ri}) +$
 - **Effektive Adresse:** $\text{EA} = (\text{Ri})$
 - **Beispiel:** $\text{INC } (\text{R0}) +$ (increment)
(Inkrementiere zunächst den Inhalt des Speicherwortes, das durch das Register R0 adressiert wird, und danach den Inhalt von R0)

Register-indirekte Adressierung

- **predecrement:** Vor der Ausführung des Befehls wird der Inhalt des Registers erniedrigt und zeigt danach auf die vorhergehende Speicherzelle
 - **Assemblerschreibweise:** $\langle \text{Mnemo} \rangle -(\text{Ri})$
 - **Effektive Adresse:** $\text{EA} = (\text{Ri}) - 1$
 - **Beispiel:** $\text{CLR} -(\text{R0})$ (clear)

(Dekrementiere zuerst den Inhalt des Registers R0 und lösche danach das Speicherwort, das durch R0 adressiert wird)

Indizierte Adressierung (*indexed addressing*)

- Die effektive Adresse wird durch die Addition des Inhalts eines Registers zu einem angegebenen Basiswert berechnet. (Adressdistanz zu einem Basiswert, Tabellenverarbeitung)

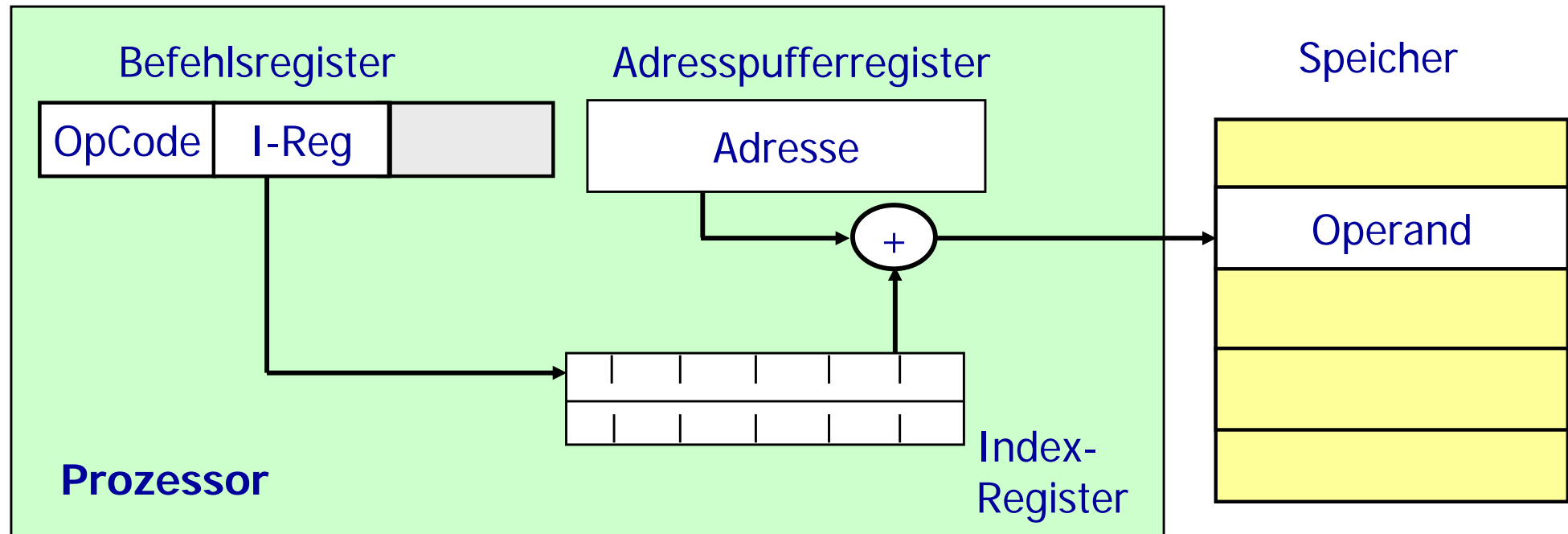
- Je nachdem, in welcher Form der Basiswert vorgegeben wird, kann man unterscheiden zwischen:
 - Speicher-relative Adressierung (memory relative addressing)
 - Register-relative Adressierung (register relative addressing)
 - Register-relative Adressierung mit Index (Based indexed mode)

Indizierte Adressierung (*indexed addressing*)

Speicher-relative Adressierung (*memory relative addressing*)

- Der Basiswert wird als absolute Adresse im Befehl vorgegeben
- **Assemblerschreibweise:** <Mnemo> <Adresse> (Ii)
- **Effektive Adresse:** $EA = ((PC) + 1) + (Ii)$

Speicher-relative Adressierung



Beispiel:

ST R1,\$A704(R0) (store)

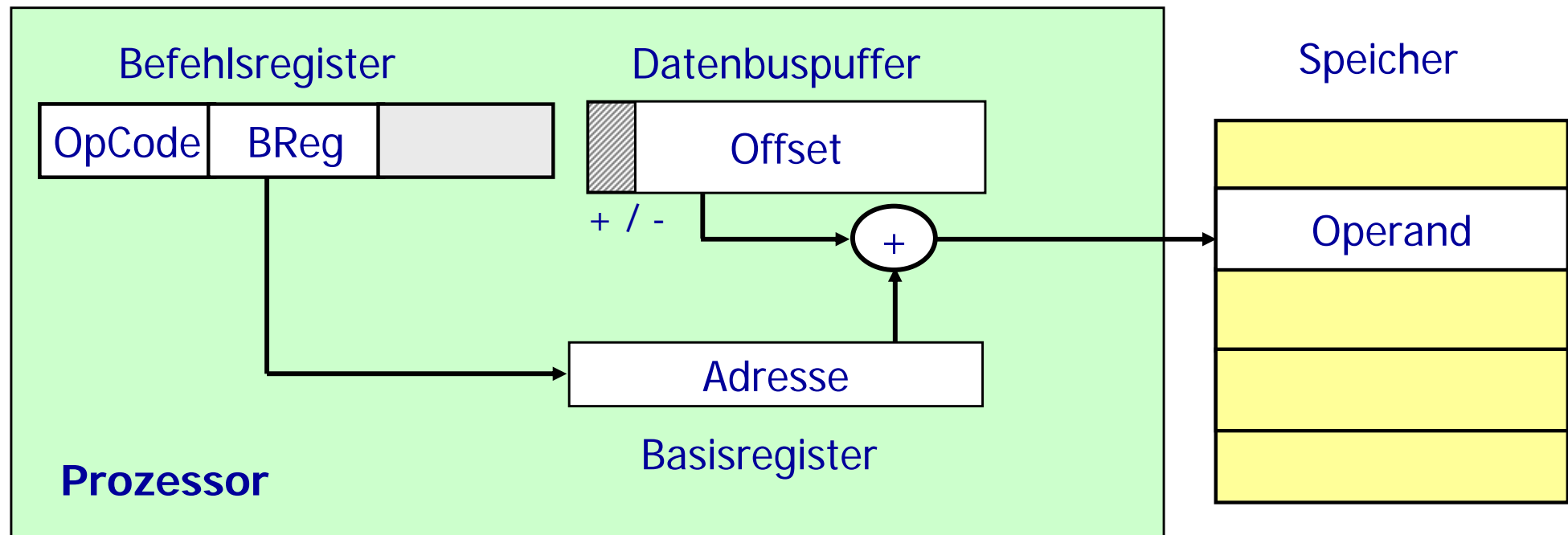
(Speichere den Inhalt von R1 in das Speicherwort, dessen Adresse sich durch Addition des Inhaltes von R0 zur Basis \$A704 ergibt)

Indizierte Adressierung (*indexed addressing*)

Register-relative Adressierung (*register relative addressing, based mode*)

- Der Basiswert befindet sich in einem Basisregister, auf das durch das BReg-Feld im OpCode verwiesen wird. Im Befehl wird ein Offset angegeben, der zum Inhalt des Basisregisters addiert wird.
- **Assemblerschreibweise:** <Mnemo> <Offset> (Bi)
- **Effektive Adresse:** $EA = (Bi) + ((PC) + 1)$

Register-relative Adressierung



Beispiel:

CLR \$A7(B0) (clear)

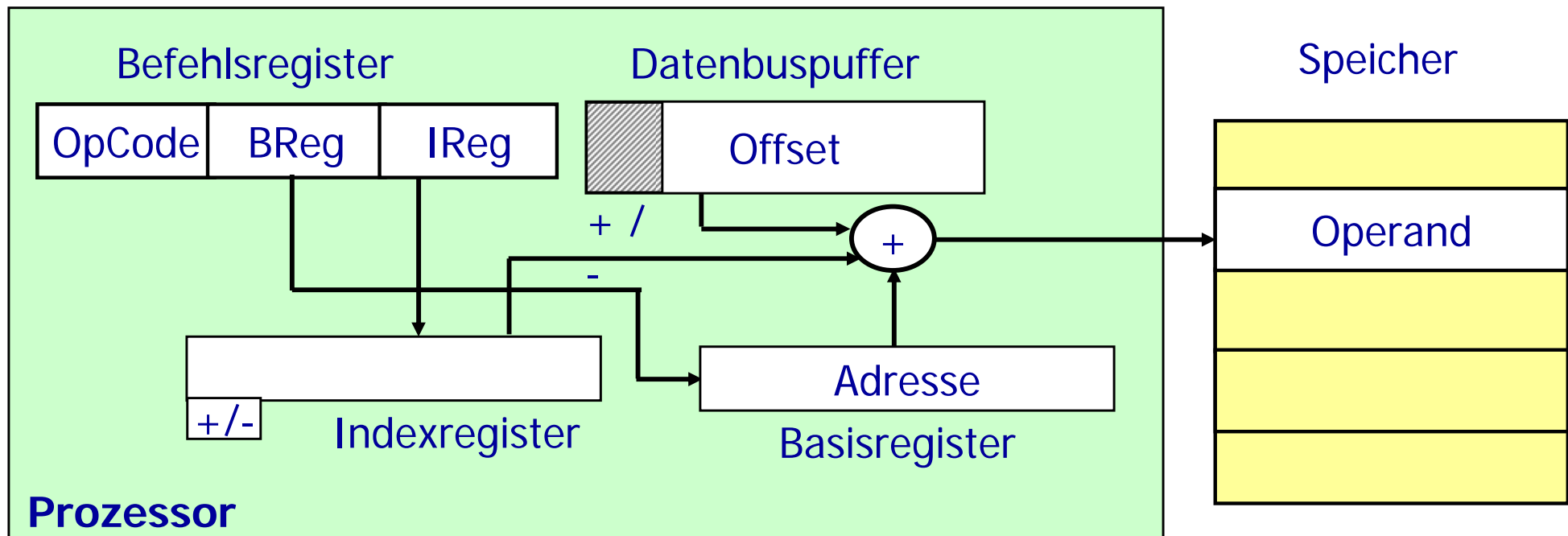
(Lösche das Speicherwort, dessen Adresse sich durch die Addition des hexadezimalen Offsets \$A7 zum Inhalt des Basisregisters B0 ergibt)

Indizierte Adressierung (*indexed addressing*)

Register-relative Adressierung mit Index (*based index mode*)

- der Basiswert wird in einem Basisregister übergeben. Dazu wird der Inhalt eines Indexregisters addiert. Für dieses Indexregister kann wieder die automatische Veränderung „autoincrement/autodecrement“ gewählt werden. Zusätzlich kann häufig im Befehl ein Offset angegeben werden, der hinzuaddiert wird.
- **Assemblerschreibweise:** <Mnemo> <Offset> (Bi)(Ii)
- **Effektive Adresse:** $EA = (Bi) + (Ii) + ((PC) + 1)$

Register-relative Adressierung mit Index



Beispiel:

DEC \$A7(B0)(I0)+ (decrement)

(Dekrementiere das Speicherwort, dessen Adresse sich durch Addition der Inhalte der Register I0 und B0 zum Offset \$A7 ergibt, und erhöhe danach den Inhalt des Registers I0)

Indizierte Adressierung (*indexed addressing*)

Befehlszähler-relative Adressierung (*program counter relative addressing*)

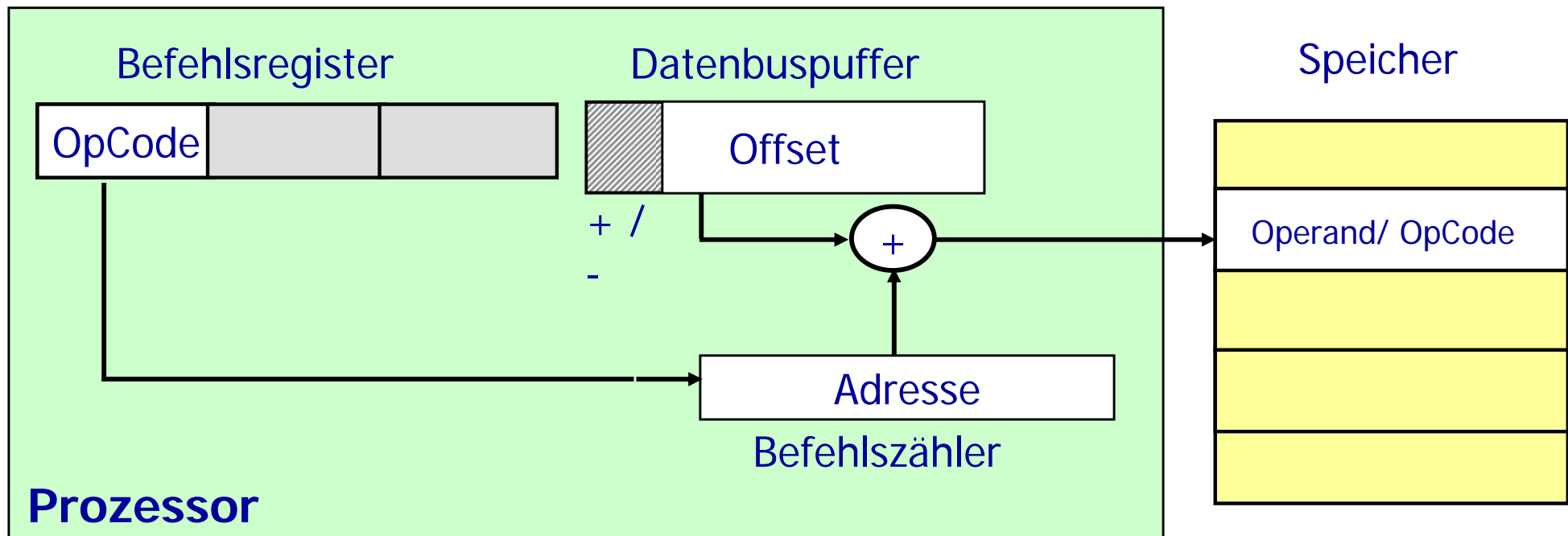
- Die effektive Adresse entsteht durch die Addition eines im Befehl angegebenen Offsets zum aktuellen Befehlszählerstand.
- Diese Adressierungsart erlaubt es, Programme im Hauptspeicher "frei" zu verschieben

- **Assemblerschreibweise:**

$\langle \text{Mnemo} \rangle \quad \langle \text{Offset} \rangle (\text{PC}) \quad \text{oder nur}$
 $\langle \text{Mnemo} \rangle \quad \langle \text{Offset} \rangle$

- **Effektive Adresse:** $EA = (\text{PC}) + 2 + ((\text{PC}) + 1)$

Befehlszähler-relative Adressierung



Beispiel:

LBRA \$7FFF (*long branch always*)

(Verzweige "unbedingt" zu der Speicherzelle, deren Adressdistanz zum aktuellen Programmzähler \$7FFF ist)

Visualisierung

- Register-Adressierung und
- Einstufige Speicher-Adressierung

Siehe TI-Homepage

<http://ti.ira.uka.de/Adressierungsarten/>

