4.4 Adressierungsarten

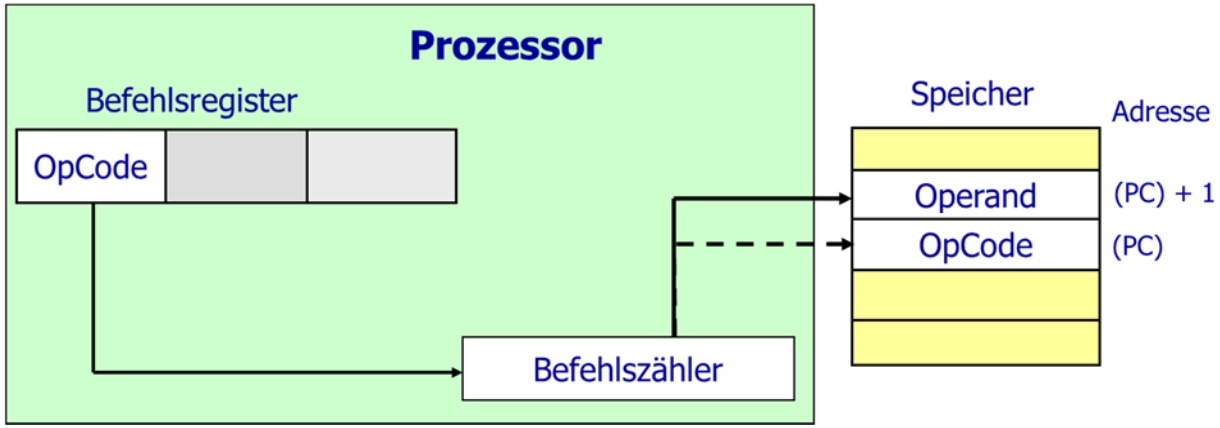
- **Einstufige Speicher-Adressierung**
  - Eine Adressberechnung zur Ermittlung der effektiven Adresse notwendig
  - Unmittelbare Adressierung (immediate addressing)
  - Direkte Adressierung (direct addressing)
    - Absolute Adressierung
    - Seiten-Adressierung
  - Register-indirekte Adressierung (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)

4.4 Adressierungsarten

- **Einstufige Speicher-Adressierung**
  - **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



Beispiel:
LDA #$A3 (load accumulator)

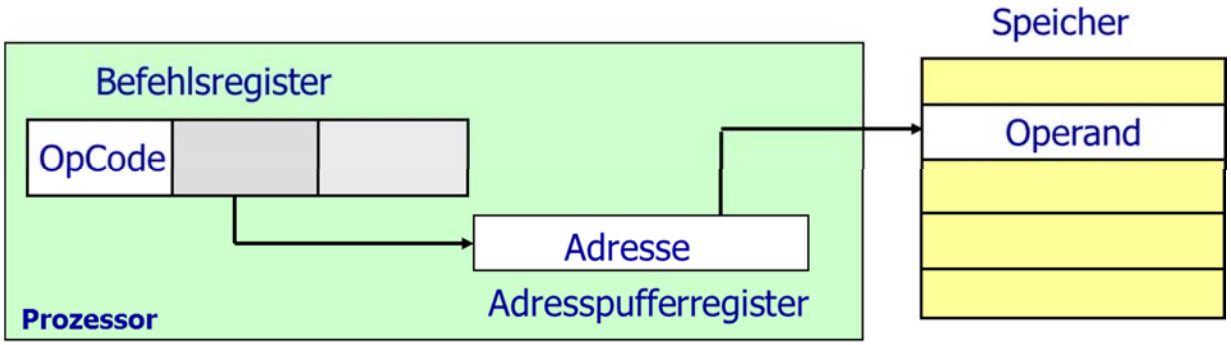
(Lade den Akkumulator A mit dem Hexadezimalwert $A3)

4.4 Adressierungsarten

- **Einstufige Speicher-Adressierung**
  - **Direkte Adressierung (direkt 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)



Beispiel:
JMP $07FE (jump)

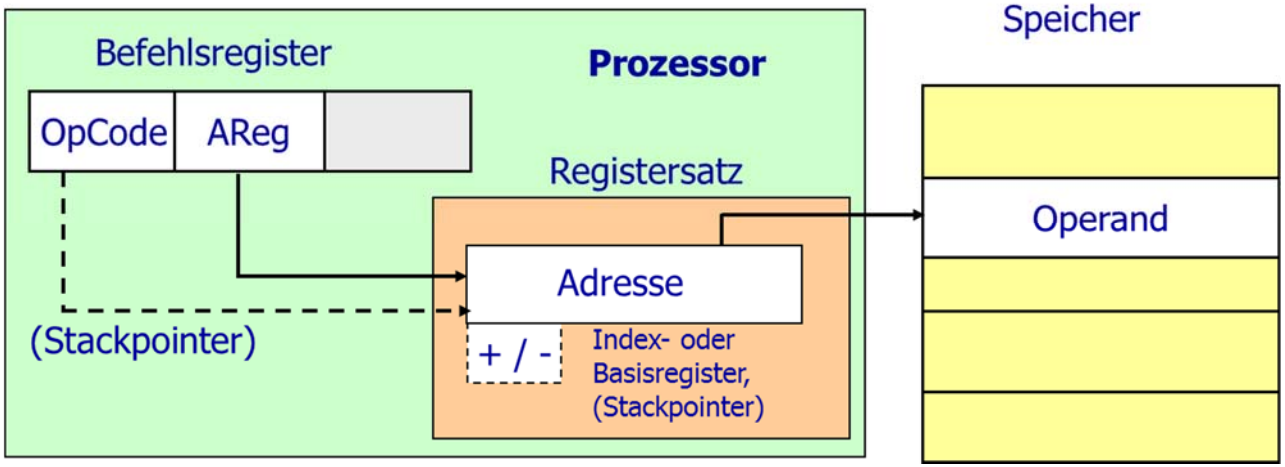
(Springe zur Adresse $07FE)

4.4 Adressierungsarten

- **Einstufige Speicher-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)



Beispiel:
LD R1, (A0) (load)

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

4.4 Adressierungsarten

- **Einstufige Speicher-Adressierung**
  - **Register-indirekte Adressierung (register indirect addressing)**
    - 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:

      <Mnemo> (Ri)+
    - Effektive Adresse:

      EA = (Ri)

Beispiel:
INC (R0)+ (increment)

(Inkrementiere zunächst den Inhalt des Speicherwortes, das durch das Register R0 adressiert wird, und danach den Inhalt von R0)

4.4 Adressierungsarten

- **Einstufige Speicher-Adressierung**
  - **Register-indirekte Adressierung (register indirect addressing)**
    - Bei der im Register stehenden Adresse handelt es sich oft um die Anfangs- oder Endadresse eines Tabellenbereichs im Speicher
      - Registerinhalt automatisch modifizieren
    - predecrement: Vor der Ausführung des Befehls wird der Inhalt des Registers erniedrigt und zeigt danach auf die vorhergehende Speicherzelle
    - Assemblerschreibweise:

      <Mnemo> -(Ri)
    - Effektive Adresse:

      EA = (Ri) – 1

Beispiel:
CLR -(R0) (clear)

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

4.4 Adressierungsarten

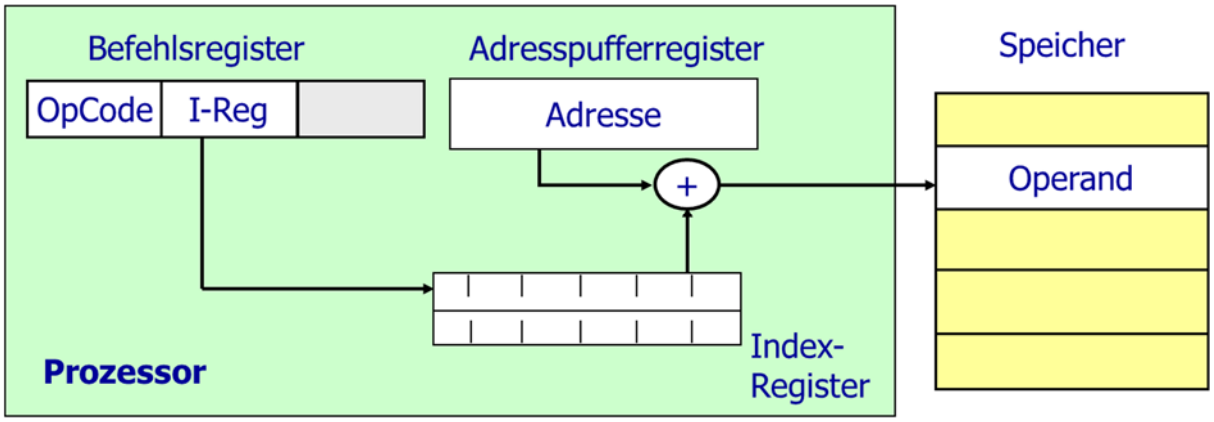
- **Einstufige Speicher-Adressierung**
  - **Indizierte Adressierung (indexed addressing)**
    - Die effektive Adresse wird durch die Addition des Inhalts eines Registers zu einem angegebenen Basiswert berechnet. (Adressdistanz zu einem Basiswert, Tabellenverarbeitung)
    - Je nachdem, in welcher Form der Basiswert vorgegeben wir, kann man unterscheiden zwischen:
      - Speicher-relative Adressierung (memory relative addressing)
      - Register-relative Adressierung (register relative addressing)
      - Register-relative Adressierung mit Index (Based indexed mode)

4.4 Adressierungsarten

- **Einstufige Speicher-Adressierung**
  - **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)



Beispiel:
ST R1,$A704(R0) (store)

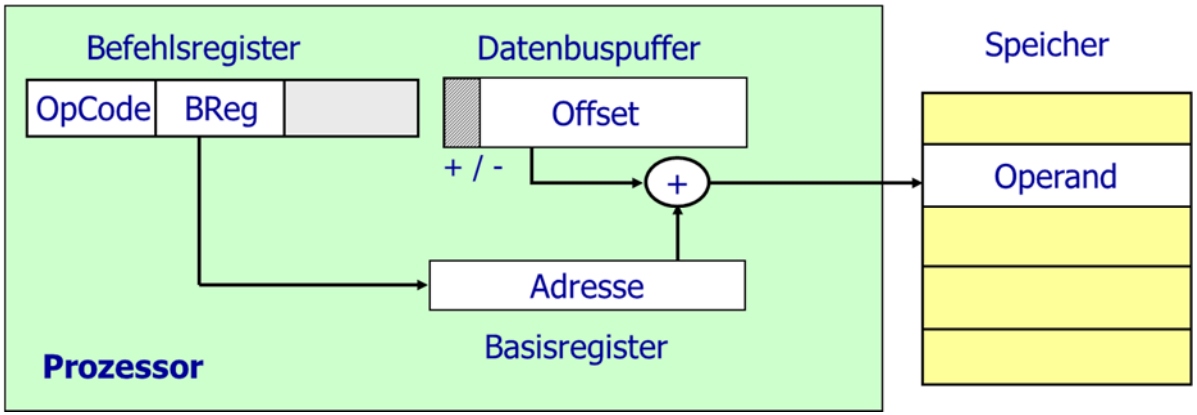
(Speichere den Inhalt von R1 in das Speicherwort, dessen Adresse sich durch Addition des Inhaltes von R0 zur Basis $A704 ergibt)

4.4 Adressierungsarten

- **Einstufige Speicher-Adressierung**
  - **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)



Beispiel:
CLR $A7(B0) (clear)

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

4.4 Adressierungsarten

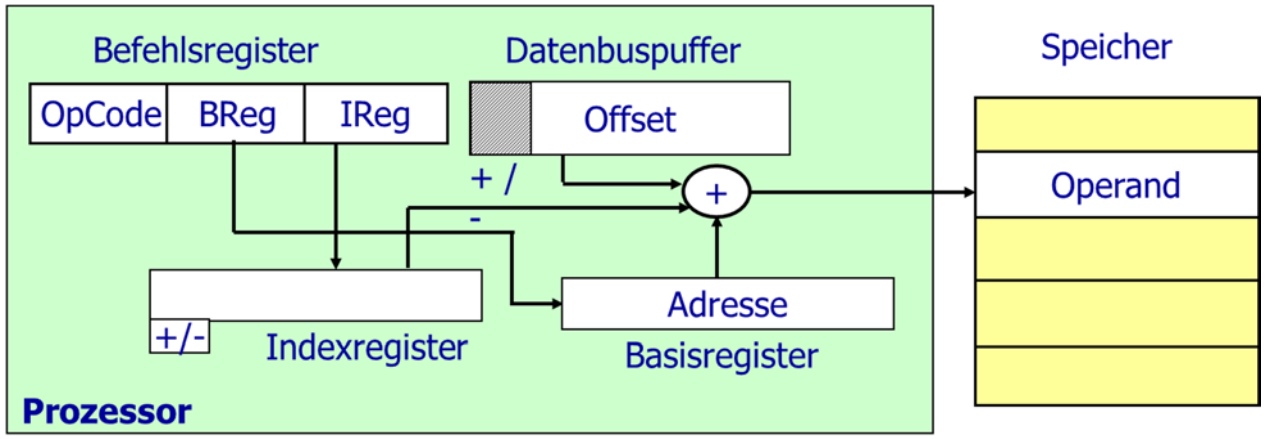
- **Einstufige Speicher-Adressierung**
  - **Register-relative Adressierung mit Index (based index mode)**
    - der Basiswert wird in einem Basisregister übergeben. Dazu wird der Inhalt eines Indexregisters addiert. Für dieses Indexregister kann wieder die automatische Veränderung „autoincrement/autodecrement“ 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)



Beispiel:
DEC $A7(B0)(I0)+ (decrement)

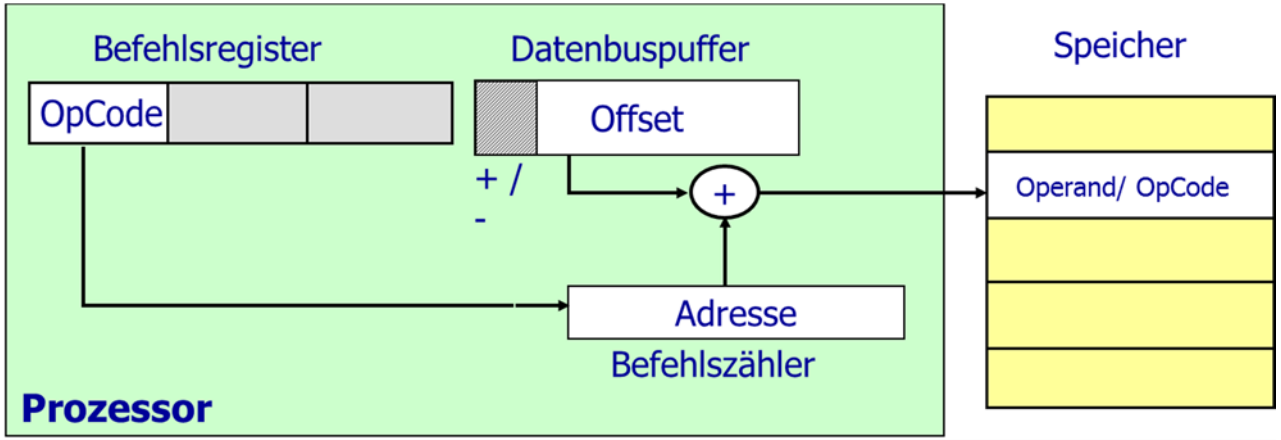
(Dekrementiere das Speicherwort, dessen Adresse sich durch Addition der Inhalte der Register I0 und B0 zum Offset $A7 ergibt, und erhöhe danach den Inhalt des Registers I0)

4.4 Adressierungsarten

- **Einstufige Speicher-Adressierung**
  - **Befehlszähler-relative Adressierung (program counter relative addressing)**
    - Die effektive Adresse entsteht durch die Addition eines im Befehl angegebenen Offsets zum aktuellen Befehlszählerstand.
    - Assemblerschreibweise:

      <Mnemo><Offset>(PC) oder nur <Mnemo><Offset>
    - Effektive Adresse:

      EA = (PC) + 2 + ((PC) + 1)



Beispiel:
LBRA $7FFF (long branch always)

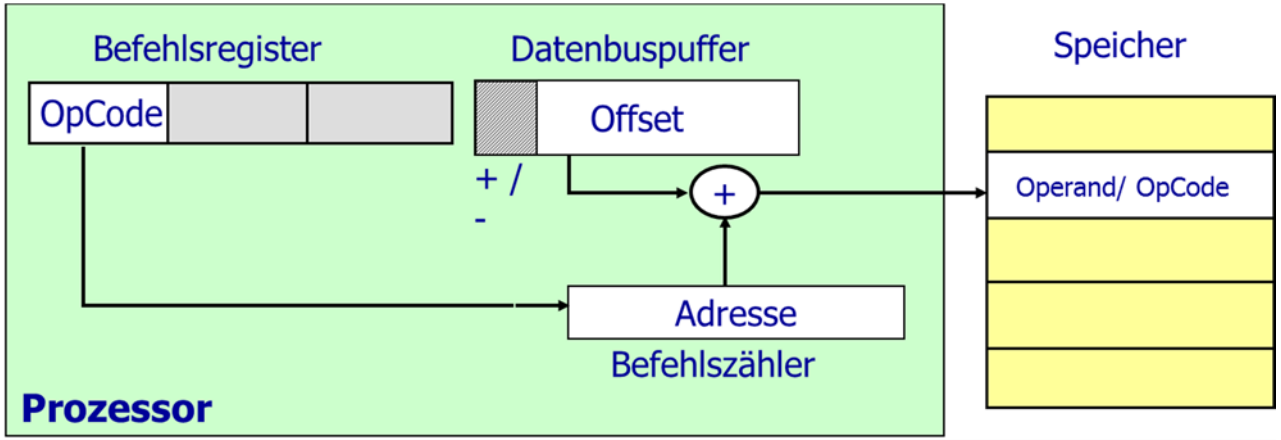
(Verzweige “unbedingt” zu der Speicherzelle, deren Adressdistanz zum aktuellen Programmzähler $7FFF ist)

4.4 Adressierungsarten

- **Einstufige Speicher-Adressierung**
  - **Befehlszähler-relative Adressierung (program counter relative addressing)**
    - Die effektive Adresse entsteht durch die Addition eines im Befehl angegebenen Offsets zum aktuellen Befehlszählerstand.
    - Assemblerschreibweise:

      <Mnemo><Offset>(PC) oder nur <Mnemo><Offset>
    - Effektive Adresse:

      EA = (PC) + 2 + ((PC) + 1)



Beispiel:
LBRA $7FFF (long branch always)

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

4.5 Befehlssatz

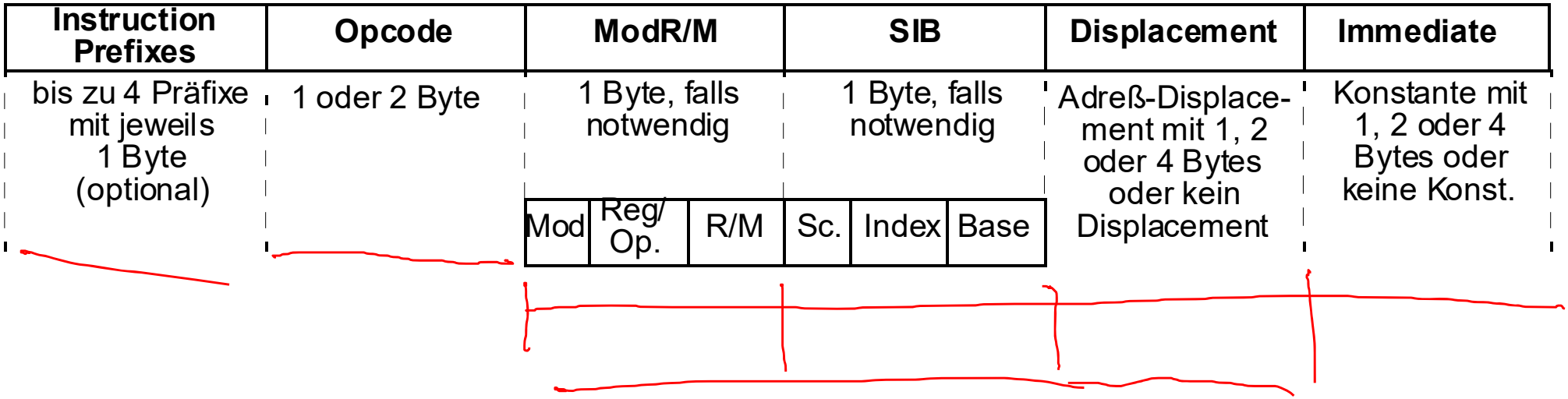
- **Der Befehlssatz (instruction set)**
  - Umfasst die Grundoperationen eines Prozessors
  - Befehlsarten:
    - Transportbefehle
    - Arithmetisch-logische Befehle
    - Schiebe- und Rotationsbefehle
    - Multimediabefehle
    - Gleitkommabefehle
    - Programmsteuerbefehle
    - Systemsteuerbefehle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

4.5 Befehlssatz

- **Das Befehlsformat**
  - Legt die Struktur des Befehls fest
    - Einem Befehlsformat liegt ein Ausführungsmodell zugrunde
    - Unterscheidung zwischen variablem und festem Befehlsformat

4.5 Befehlssatz

- **Fallstudie IA-32 Befehlsformat**
  - Befehlsformate unterschiedlicher Länge (variables Befehlsformat)
  - Byte-Struktur:
    - Länge der Befehlsformate in Vielfachen von Bytes, wobei die Länge von der Anzahl der Adressen und Adressierungsarten abhängt.



4.5 Befehlssatz

- **Fallstudie: MIPS Befehlsformat**
  - Befehlsformate mit fester Länge (festes Befehlsformat)
  - Register-Register-Befehle (Typ R)

opcode	rs	rt	rd	shamt	funct
31-26	25-21	20-16	15-11	10-6	5-0

  - Lade/Speicher Befehle (Typ I)

opcode	rs	rt	Adresse
31-26	25-21	20-16	15-0

  - Sprungbefehle (Typ J)

opcode	rs	rt	Adresse
31-26	25-21	20-16	15-0

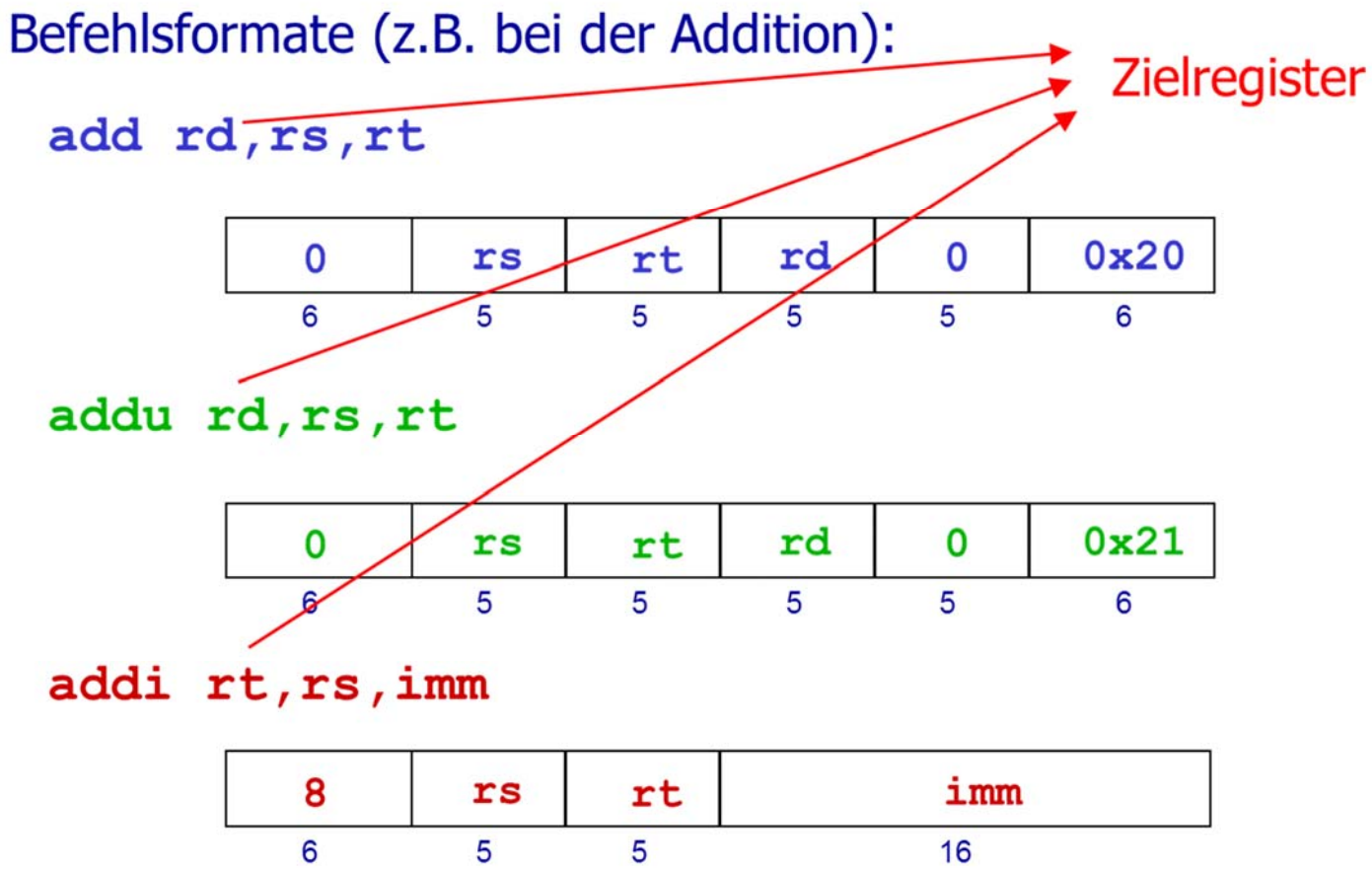
4.5 Befehlssatz

- **Fallstudie: MIPS Befehlsformat**
  - Befehlsformate mit fester Länge (festes Befehlsformat)

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 *(shift amount)*
funct	6 Bit Kodierung der Funktion *(function)*

4.5 Befehlssatz

- **Fallstudie: MIPS Befehlsformat**
  - Beispiel

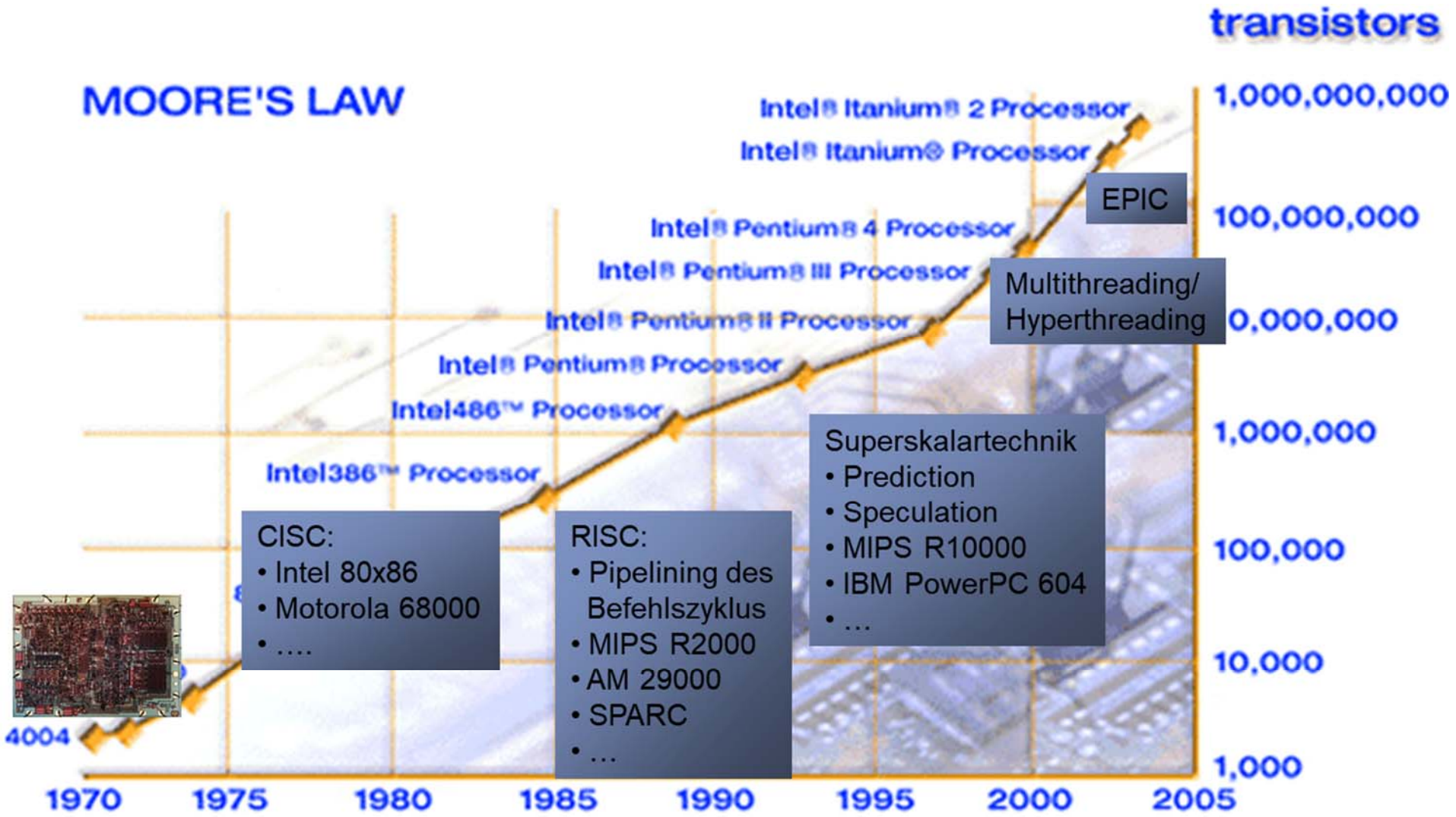


4.6 CISC vs. RISC

- **CISC Befehlssatz-Architektur (ISA)**
  - Complex Instruction Set Computer
- **RISC Befehlssatz-Architektur (ISA)**
  - Reduced Instruction Set Computer

CISC vs. RISC

- **Entwicklung der Mikroprozessoren**



4.6 CISC vs. RISC

- **CPU-Zeit**
  - Ausführungszeit eines Programms auf der CPU (gemessen in Taktzyklen oder in Zeiteinheiten)

$$CPU - Zeit = IC * CPI * Taktdauer$$

$$CPU - Zeit = \frac{IC * CPI}{Taktfrequenz}$$

  - Mit
    - IC: Anzahl der ausgeführten Befehle (Instruction Count)
    - CPI: Anzahl der Taktzyklen pro Befehl

Cycles per Instruction

$$CPI = \frac{Anzahl\ der\ Taktzyklen}{IC}$$

4.6 CISC vs. RISC

- **CISC ISA**
  - Befehlssatz mit komplexen Maschinenbefehlen
    - Entwurf der CISC Befehlssatz-Architekturen in einer Zeit, in der viel in Assembler programmiert worden ist
    - Tendenz komplexe Funktionalität mit den Instruktionen bereitzustellen, um Assembler-Programmierer zu unterstützen
    - Operationen einer Instruktion möglichst nahe an Anweisungen einer höheren Programmiersprache
      - Beispiel: Intel iAPX32 (~Mitte der 1980er Jahre):
        - Befehlssatz mit objektorientierter Verarbeitung in HW
        - Neben grundlegenden Datentypen auch Bitstreams, Arrays und Objects
  - Mikroprogrammierte Implementierung des Steuerwerks
  - Variables Befehlsformat und Befehlslänge

4.6 CISC vs. RISC

- **CISC ISA**
  - Ziel: $CPU - Zeit = IC * CPI * Taktdauer$
    - Reduzierung IC, also die Anzahl der auszuführenden Befehle für ein auszuführendes Programm
    - Je kompakter der Befehlsstrom, desto besser die Nutzung des Haupt- / Cache-Speichers, desto weniger Befehle müssen geholt werden, um ein Programm auszuführen
  - Beispiele:
    - IBM System/360, System/370 (Familienkonzept)
    - DEC VAX-11
    - Intel x86 ISA (IA-32)
    - Motorola 68000 Familie
    - …

4.6 CISC vs. RISC

- **RISC ISA**
  - Ziel: $CPU - Zeit = IC * CPI * Taktdauer$
    - Reduzierung CPI, also die Anzahl der Zyklen pro Instruktion (CPI ~ 1)
  - Beispiele:
    - Forschung bei IBM, Standford (MIPS) Berkeley (RISC)
    - AMD 29000
    - MIPS R2000, Nachfolger
    - SPARC
    - ARM
    - IBM POWER

4.6 CISC vs. RISC

- **RISC ISA**
  - Einfache Maschinenbefehle
    - Einheitliches und festes Befehlsformat
  - Load/Store Architektur
    - Befehle arbeiten auf Registeroperanden
    - Lade- und Speicherbefehle greifen auf Speicher zu
  - Einzyklus-Maschinenbefehle
    - Effizientes Pipelining des Maschinenbefehlszyklus
    - Einheitliches Zeitverhalten der Maschinenbefehle, wovon nur Lade- und Speicherbefehle sowie die Verzweigungsbefehle abweichen
  - Optimierende Compiler
    - Reduzierung der Befehle im Programm