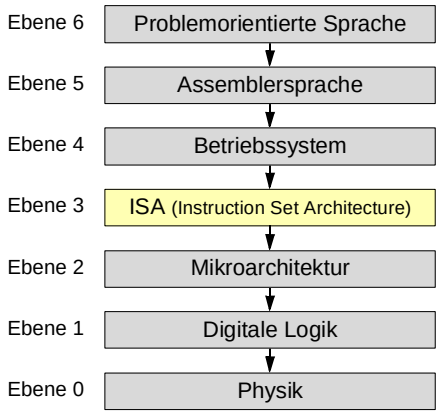
C 6809 Prozessor

1 Einordnung



2 Motivation

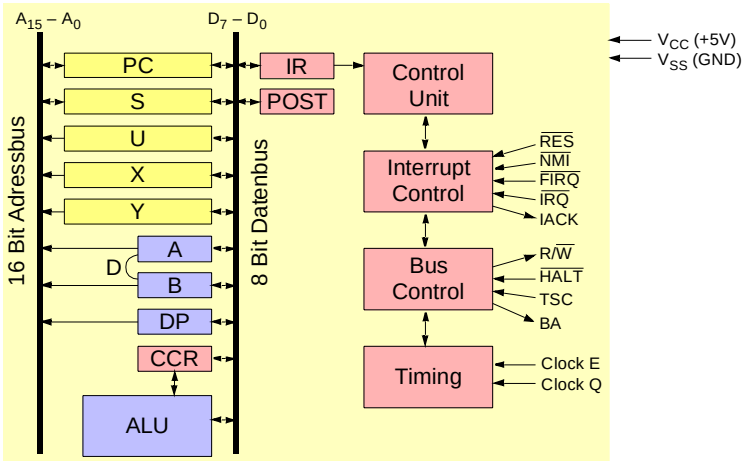
- Nachteile des hypothetischen Prozessors aus TI1
  - ein Datenregister erfordert häufiges Speichern von Zwischenergebnissen
  - kein einfacher Unterprogrammaufruf
  - kein indizierter Zugriff auf Felder möglich (z.B. **`a[i]`**)
  - Realisierung von Zeigern nahezu unmöglich
  - schmaler Satz arithmetischer Operationen
    (keine Fließkommaartihmetik, keine Multiplikation und Division)
  - keine Unterbrechungen *(Interrupts)*
  - Wortbreite n > Adressbreite b:
    Adressraum ist stark begrenzt oder Worbreite sehr groß
  - nur eine Bedingung abprüfbar (**`JMA`**)

3 Motorola 6809 Überblick

- Hersteller: Motorola, Hitachi
  - trat Erbe der 6502, 6800 und 6802 Prozessoren an
- Aufbau
  - 8 Bit-Prozessor
  - 8 Bit breiter Datenbus
  - 16 Bit breiter Adressbus (64kB Speicher adressierbar)
  - zwei 8 Bit Datenregister (A und B),
    koppelbar zu einem 16 Bit Datenregister (D)
  - zwei 16 Bit Stapelzeiger/Stackpointer (S und U)
  - zwei 16 Bit Indexregister (X und Y)
  - Unterstützung für Unterbrechungen
  - erster Mikroprozessor mit 8x8→16 Bit Multiplikationsbefehl

3 Motorola 6809 Überblick (2)

- Blockschaltbild

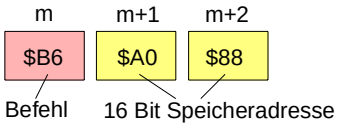


3 Motorola 6809 Überblick (3)

- Instruktionen
  - 59 Maschinenbefehle
  - 10 Adressierungsarten (mit 24 Unterarten)
    →1464 unterschiedliche Operationen
  - weitgehend orthogonaler Befehlssatz
- Befehlsklassen
  - Datentransferbefehle
  - ALU-Befehle
  - Sprungbefehle
  - Stack- und Index-Befehle
  - Systembefehle

3 Motorola 6809 Überblick (4)

- Befehlskodierung
  - wichtige Befehle in einem Byte
  - für einige Befehle ist Zusatzbyte erforderlich
    - Postbyte für einige Adressierungsarten
    - Prebyte für manche Befehle
  - Speicheradressen folgen in weiteren Bytes (byte-orientierter Prozessor)



Beispiel: **`LDA $A088`**

lädt Register A mit Inhalt
von Speicherstelle $A088

3 Motorola 6809 Überblick (5)

- Statusregister (CCR, Condition-Code-Register)
  - 8-Bit-breites Register
  - Einzelbits mit unterschiedlicher Bedeutung
    - Kontrollbits zur Steuerung des Mikroprozessors
    - Statusbits zur Anzeige von bestimmten Zuständen
  - Beispiel: Carry-Flag (**C**)
    - typische Bedeutung: Ergebnis des Carry bei Addition
      – Überlauf bei vorzeichenloser Addition
    - weitere Bedeutungen:
      – Eingabewert für ALU als Carry-Wert bei Addition
      – Carry für Subtraktion
      – Zwischenpuffer für Schiebeoperationen
  - Statusbits werden bei einigen Befehlen implizit gesetzt oder gelöscht

4 Befehlssatz

4.1 Datentransferbefehle

- Unterscheidung von Lade- und Speicherbefehlen sowie Transferbefehlen
  - Ladebefehle (lädt Wert aus Speicher in Register)

Mnemonic	Beschreibung
LDA, LDB	Lade 8 Bit Register
LDD, LDS, LDU, LDX, LDY	Lade 16 Bit Register

    - letzter Buchstabe bezeichnet Register

  - Speicherbefehle (speichert Wert vom Register in Speicher)

STA, STB	Speichere 8 Bit Register
STD, STS, STU, STX, STY	Speichere 16 Bit Register

4.1 Datentransferbefehle (2)

- Transferbefehle (Transfer zwischen Registern)

Mnemonic	Beschreibung
EXG	Tausche zwei Registerinhalte
TFR	Transferiere von einem zum anderen Register

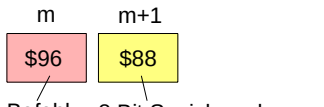
  - Tausch und Transfer immer nur zwischen 8 Bit- bzw. 16 Bit-Registern
  - beliebige Registerkombinationen möglich (auch PC, CC und DP)

4.2 Einfache Adressierungsarten

- Speicheradressierung 16 Bit *(Extended)*
  - Speicheradresse wird mit zwei Bytes hinter dem Befehl angegeben
  - z.B. **LDA $A088** oder **STX $A08A**
  - bei 16 Bit-breiten Operationen wird angegebene Adresse und Folgeadresse verwendet
  - 16 Bit-Werte werden MSB:LSB gespeichert *(Big-Endian-Format)*
- Unmittelbare Adressierung *(Immediate)*
  - Wert wird hinter dem Befehl unmittelbar angegeben (nicht bei Speicherbefehlen)
  - z.B. **LDA #$35** oder **LDX #$1234**
  - Wert wird hier in das Register geladen

4.2 Einfache Adressierungsarten (2)

- Speicheradressierung 8 Bit *(Direct)*
  - arbeitet auf Speicherstelle im Speicher
  - oberen 8 Bit (MSB) der Adresse kommen aus dem DP-Register *(Direct-Page)*
  - unteren 8 Bit (LSB) der Adresse werden hinter dem Befehl gespeichert



Beispiel: **LDA $88**

lädt Register A mit Inhalt von Speicherstelle $xy88 mit DP = $xy

- ★ Sinn
  - Einsparung von Speicher für die Maschinenprogramme bei häufig zugegriffenen Speicherstellen (z.B. für wichtige Variablen)

4.3 ALU-Befehle

- Arithmetische Operationen

Mnemonic	Beschreibung
ADDA, ADDB, ADDD	Addiere Speicherwert zu Register A, B oder D
ADCA, ADCB	Addiere Speicherwert plus Carry zu Register A oder B
SUBA, SUBB, SUBD	Subtrahiere Speicherwert von Register A, B oder D
SBCA, SBCB	Subtrahiere Speicherwert mit Carry von Register A oder B
MUL	Multipliziere A x B, Ergebnis nach D (vorzeichenlose Darstellung)
INC, INCA, INCB	Inkrementiere Speicherstelle oder Register A oder B
DEC, DECA, DECB	Dekrementiere Speicherstelle oder Register A oder B

4.3 ALU-Befehle (2)

- Addition und Carry
  - Carry-Flag wird als Ergebnis der Addition gesetzt
  - Addition von großen Zahlen (z.B. 24 Bit)
    - Addition der niederwertigsten 8 Bit (z.B. **ADDA**)
    - Addition der nächst-höherwertigen 8 Bit mit Einbezug des Carry-Ergebnisses von vorher (z.B. **ADCA**)
    - usw.

4.3 ALU-Befehle (3)

- Subtraktion und Carry
  - M6809 arbeitet im Zweierkomplement
  - Subtraktion ist Addition des Einerkomplements plus 1
  - Carry als Ergebnis
    - 0 = kein Unterlauf
    - 1 = Unterlauf, „Eins“ geborgt
  - Carry als Eingabewert
    - 0 = normale Subtraktion
    - 1 = eine Eins geborgt von vorheriger Subtraktion (bei Subtraktion langer Zahlen), d.h. Ergebnis ist um Eins kleiner
- In der ALU: Carry bei Subtraktion invertiert

4.3 ALU-Befehle (4)

- Logische Operationen

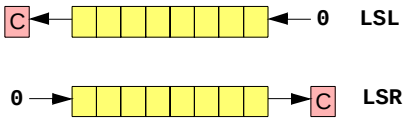
Mnemonic	Beschreibung
ANDA, ANDB, ANDCC	Logisches UND von Speicherwert und Register (**ANDCC** nur mit Immediate-Adressierung)
ORA, ORB, ORCC	Logisches ODER von Speicherwert und Register (**ORCC** nur mit Immediate-Adressierung)
EORA, EORB	Logisches XOR von Speicherwert und Register
NEG, NEGA, NEGB	Zweierkomplement in Speicherstelle oder Register A oder B bilden
COM, COMA, COMB	Einerkomplement in Speicherstelle oder Register A oder B bilden

4.3 ALU-Befehle (5)

- Schiebeoperationen

Mnemonic	Beschreibung
LSL, LSLA, LSLB	Logisches Schieben nach links (Null einschieben) einer Speicherstelle oder des Registers A oder B
LSR, LSRA, LSRB	Logisches Schieben nach rechts (Null einschieben) einer Speicherstelle oder des Registers A oder B

  - Carry-Flag enthält herausgeschobenes Bit

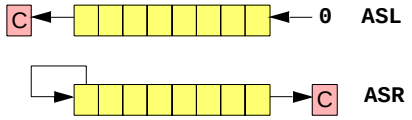


4.3 ALU-Befehle (5)

- Schiebeoperationen

Mnemonic	Beschreibung
ASL, ASLA, ASLB	Arithmetisches Schieben nach links (Null einschieben) einer Speicherstelle oder des Registers A oder B
ASR, ASRA, ASRB	Arithmetisches Schieben nach rechts (Vorzeichen einschiben) einer Speicherstelle oder des Registers A oder B

  - Carry-Flag enthält herausgeschobenes Bit

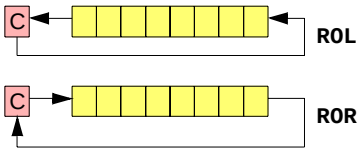


4.3 ALU-Befehle (6)

- Schiebeoperationen (fortges.)

Mnemonic	Beschreibung
ROL, ROLA, ROLB	Rotieren über Carry nach links einer Speicherstelle oder des Registers A oder B
ROR, RORA, RORB	Rotieren über Carry nach rechts einer Speicherstelle oder des Registers A oder B

  - Carry-Flag wird eingeschoben
  - Carry-Flag erhält ausgeschobenes Bit



4.3 ALU-Befehle (7)

- Test-, Vergleichsoperationen u.a.

Mnemonic	Beschreibung
TST, TSTA, TSTB	Vergleiche Speicherwert oder Register A oder B mit Null (setze CCR)
BITA, BITB	Teste bitweise Speicherwert mit Register A oder B (UND-Verknüpfung)
CMPA, CMPB, CMPD, CMPS, CMPU, CMPX, CMPY	Subtrahiere Speicherwert von Register (A, B, D, S, U, X oder Y) jedoch ohne Speichern des Ergebnisses (setze CCR)
CLR, CLRA, CLRB	Lösche Speicherstelle oder Register A oder B (speichere Null)
SEX	Fülle A mit Vorzeichenbits von B (Sign-Extension)
DAA	Berrichtigung des Registers A für BCD-Addition (Decimal Adjust)

4.4 Sprungbefehle

- Relative Sprünge
  - Sprungadresse ist ein Versatz im Zweierkomplement zum aktuellen Programmzähler, z.B. **+46**: springt an die Stelle **PC+46**
  - 8 Bit Versatz: –128 bis +127 *(Branch)*
  - 16 Bit Versatz: –32768 bis +32767 *(Long Branch)*
  - Überlauf wird ignoriert, alle Speicherstellen erreichbar
- Unbedingte Sprünge

Mnemonic	Beschreibung
BRA, LBRA	*Branch Always*, springe immer
BRN, LBRN	*Branch Never*, springe niemals

JMP	*Jump*, springe immer (absolute Adressangabe sowie mehrere Adressierungsarten)

4.4 Sprungbefehle (2)

- Bedingte Sprünge über Condition-Flags

Mnemonic	Beschreibung
BCC a	*Branch Carry Clear*, springe bei C = 0
BCS a	*Branch Carry Set*, springe bei C = 1
BNE a	*Branch Not Equal*, springe bei Z = 0
BEQ a	*Branch Equal*, springe bei Z = 1
BPL a	*Branch Plus*, springe bei N = 0
BMI a	*Branch Minus*, springe bei N = 1
BVC a	*Branch Overflow Clear*, springe bei V = 0
BVS a	*Branch Overflow Set*, springe bei V = 1

4.4 Sprungbefehle (3)

- Bedingte Sprünge bei vorzeichenbehafteten Vergleichen/Subtraktionen

Mnemonic	Beschreibung
BGT, LBGT	*Branch Greater Than*, springe falls größer $(N \cdot V \cdot \bar{Z} + \bar{N} \cdot \bar{V} \cdot \bar{Z} = 1)$
BGE, LBGE	*Branch Greater Equal*, springe falls größer gleich $(N \cdot V + \bar{N} \cdot \bar{V} = 1)$
BLT, LBLT	*Branch Less Than*, springe falls kleiner $(N \cdot \bar{V} + \bar{N} \cdot V = 1)$
BLE, LBLE	*Branch Less Equal*, springe falls kleiner gleich $(Z + N \cdot \bar{V} + \bar{N} \cdot V = 1)$

- identisch mit Hypothetischem Prozessor

4.4 Sprungbefehle (4)

- Bedingte Sprünge bei vorzeichenlosen Vergleichen/Subtraktionen

Mnemonic	Beschreibung
BHI, LBHI	*Branch High*, springe falls größer $(\bar{C} \cdot \bar{Z} = 1)$
BHS, LBHS	*Branch High or Same*, springe falls größer gleich $(\bar{C} = 1)$
BLO, LBLO	*Branch Low*, springe falls kleiner $(C = 1)$
BLS, LBLS	*Branch Low or Same*, springe falls logisch kleiner oder gleich $(C + Z = 1)$

- **BHS** entspricht **BCC**, **BLO** entspricht **BCS**

4.5 Statusregister

- Flags im Statusregister (Condition-Code-Register)

Bit 7 Bit 0

E	F	H	I	N	Z	V	C

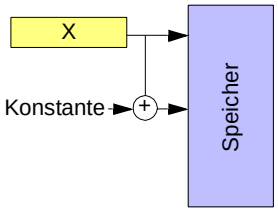
  - **Carry:** Carry bei Addition, Borrow bei Subtraktion/Vergleich (inverses Carry eines Standardaddierers)
  - **Overflow:** Überlauf bei vorzeichenbehafteten Operationen
  - **Zero:** Ergebnis gleich Null
  - **Negative:** Ergebnis negativ
  - **Interrupt:** Maskierung der Unterbrechung IRQ (I = 1: gesperrt)
  - **Half Carry:** Carry von Bit 3 auf Bit 4 (für BCD Arithmetik notwendig, Befehl **DAA**)
  - **Fast Interrupt:** Maskierung der Unterbrechung FIRQ (F = 1: gesperrt)
  - **Entire Flag:** alle Register auf Stack

4.6 Indizierte Adressierungsarten

- Viele Befehle erlauben weitere Adressierungsarten *(Indexed)*
  - Postbyte hinter der Befehlscodierung gibt genauen Typ an
  - Typen:
    - konstant indiziert
    - registerindiziert
    - indiziert mit Autoinkrement oder Autodekrement
    - PC-relativ
  - fast alle Typen können noch indirekt benutzt werden

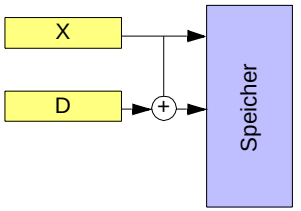
4.6 Indizierte Adressierungsarten (2)

- Konstant indizierte Adressierung
  - Konstante = 0
    - Beispiel: **`LDA ,X`**
    - Speicheradresse wird aus einem Adressregister (hier X) genommen
    - Register A wird mit Speicherinhalt aus der von X adressierten Speicherstelle geladen
    - mögliche Adressregister: X, Y, U, S
  - Konstante ≠ 0
    - Beispiel: **`LDA 3,X`** bzw. **`LDA -3,X`**
    - Speicheradresse wird durch Addition der Konstante mit dem Register ermittelt
    - Konstante vorzeichenbehaftet
    - 5 Bit, 8 Bit oder 16 Bit breite Konstante



4.6 Indizierte Adressierungsarten (3)

- Registerindizierte Adressierung
  - Speicheradresse ergibt sich aus Addition von Datenregister und Adressregister
  - Beispiele: **`LDA B,X`** oder **`LDA D,X`**
  - Datenregister A und B: 8 Bit Versatz vorzeichenbehaftet
  - Datenregister D: 16 Bit Versatz vorzeichenbehaftet



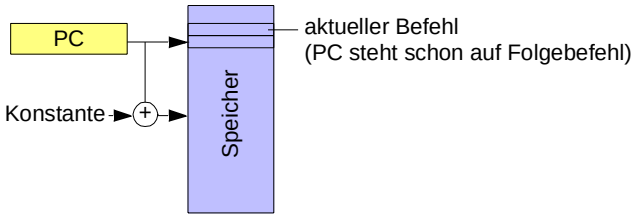
- ★ Einsatz
  - Adressregister zeigt auf Datenstruktur im Speicher
  - variabler oder konstanter Versatz selektiert Komponente der Datenstruktur

4.6 Indizierte Adressierungsarten (4)

- Indizierung mit Autodekrement oder Autoinkrement
  - identisch mit konstanter Indizierung mit Null
  - zusätzlich Dekrementierung oder Inkrementierung des Adressregisters
  - Beispiel: **`LDA ,-X`** oder **`LDD ,--Y`**
  - Dekrementierung vor Berechnung der effektiven Adresse zum Speicherzugriff
  - Dekrementierung um 1 oder um 2
  - Beispiel: **`LDA ,X+`** oder **`LDD ,Y++`**
  - Inkrementierung nach Berechnung der effektiven Adresse zum Speicherzugriff
  - Inkrementierung um 1 oder um 2

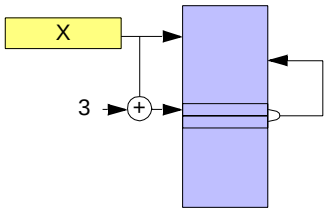
4.6 Indizierte Adressierungsarten (5)

- PC-relative Adressierung *(Relative)*
  - Speicheradresse wird relativ zum aktuellen Programmzählerstand gespeichert (ähnlich wie Sprungadressen relativer Sprungbefehle)
  - Versatz 8 Bit oder 16 Bit breit
  - Beispiel: **`LDA $54,PCR`** bzw. **`LDA –$21c0,PCR`**



4.7 Indirekte Adressierungsarten

- Indirekte indizierte Adressierungsarten
  - alle indizierten Adressierungsarten können indirekt erfolgen, d.h. effektive Adresse steht im Speicher
  - Beispiel: **`LDA [3,X]`**



Adresse wird mit MSB zuerst gespeichert

- ★ Einsatz
  - in der Datenstruktur, auf die X zeigt, steht ein Zeiger auf ein Datum
  - nur ein Befehl zum Zugriff nötig

4.7 Indirekte Adressierungsarten (2)

- Indirekte indizierte Adressierung (fortges.)
  - Weitere Beispiele:
    - indirekt registerindiziert: **`LDA [B,Y]`**
    - indiziert mit Autodekrement/Autoinkrement: **`LDA [,U++]`** (De- bzw. Inkrement nur um 2 möglich, da Adresse zwei Byte lang)
    - indirekt PC-relativ: **`LDD [$40,PCR]`**
- Indirekte absolute Adressierung *(Indexed Extended)*
  - Angabe einer 16 Bit Speicheradresse bei der eine Adresse im Speicher steht
  - Beispiel: **`LDA [$34FE]`**