Einstieg in die Mikrocontrollertechnik

Modul 12235

Dirk Killat

Inhalt

Elektrotechnik & Informationstechnik b-tu
Einstieg in die Mikrocontrollertechnik
Lehrstuhl Mikroelektronik Prof. Dr.-Ing. D. Killat

Elektrotechnik & Informationstechnik b-tu
Einstieg in die Mikrocontrollertechnik
Lehrstuhl Mikroelektronik Prof. Dr.-Ing. D. Killat

Elektrotechnik & Informationstechnik b-tu
Einstieg in die Mikrocontrollertechnik
Lehrstuhl Mikroelektronik Prof. Dr.-Ing. D. Killat

1 Einleitung

Wir stellen uns die Frage nach dem Warum, d.h. was hat zur Entwicklung des Mikrocontrollers geführt (Motivation) und definieren den Begriff *Mikrocontroller*.

- Abgrenzung Mikroprozessor und Mikrocontroller
- Mikrocontroller als mathematischer Rechner
- Mikrocontroller als universeller programmierbarer Automat
- Meilensteine der Mikroprozessortechnik

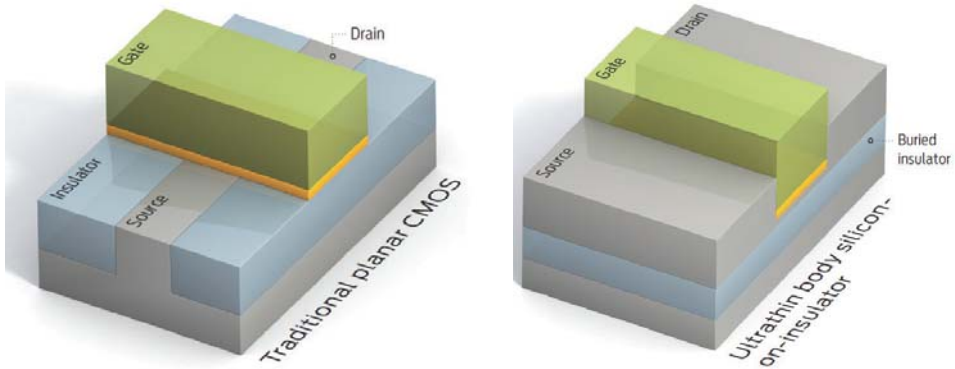
1.1 Motivation für die Entwicklung

- Durchführung wiederholter Berechnungen
- Universeller programmierbarer Automat

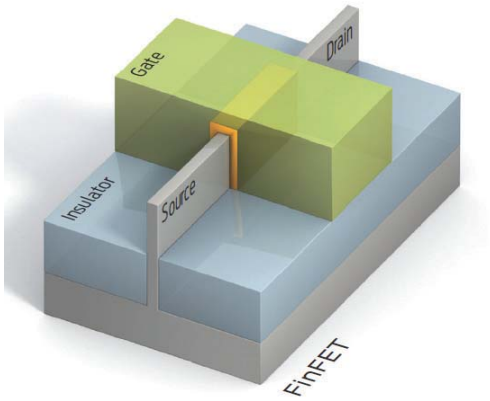
1.2.4 Das erste Mikroprozessorsystem auf Silizium

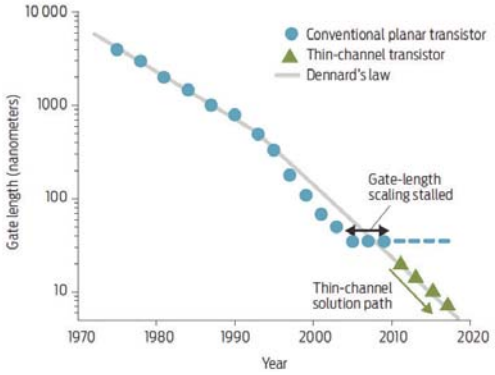
Quelle: Wikipedia Commons, aus: Journal Electronic News, 1971

1.3.3 Nano-Technologien



Quelle: IEEE Spectrum, Vol. 48, Issue 11, Nov. 2011, Ahmed: Transistor Wars



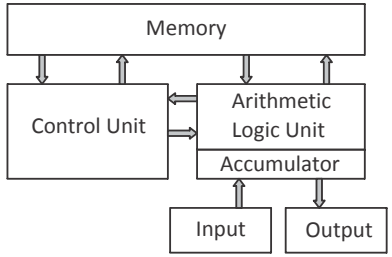


3 Der einfachste Mikroprozessor MU0

- Der „MU0“ ist ein hypothetischer Mikroprozessor, der dem an der Universität Manchester 1948 entwickelten Small-Scale Experimental Machine (1948) entspricht.
- Programm und Daten sind in einem gemeinsamen Speicher untergebracht: von Neumann – Architektur
- Der MU0 wird weltweit zu Lehrzwecken eingesetzt.

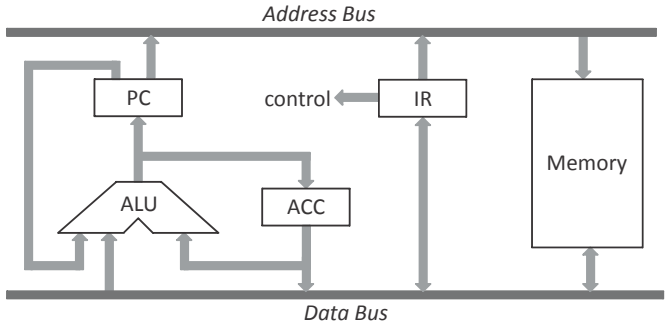
3.1 Aufbau eines Rechnersystems

- Speicher (Memory) für Programm und Daten
- Arithmetisch-Logische Einheit (Rechenwerk) „ALU“ für Berechnungen
- Ergebnisse aus der ALU werden im Akkumulator (kurz: „Akku“) gespeichert
- Ein- und Ausgabe der zu verarbeitenden Daten
- Steuerwerk (Control Unit) interpretiert Befehle und steuert den Programmablauf



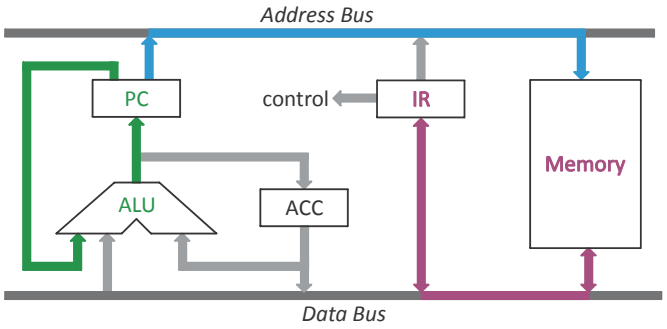
3.2 Aufbau des MU0-Rechners

- Neben der Arithmetisch-Logischen Einheit (ALU), dem Akkumulator-Register (ACC) und dem Speicher (Memory) benötigt unser MU0 noch:
  - Programmzähler (Program Counter, PC)
  - Befehlsregister (Instruction Register, IR)
  - Bus mit Adress-Leitungen zum Speicher (Address Bus)
  - Bi-direktionaler Bus mit Daten-Leitungen zwischen ALU, Akku, Befehlsregister und Speicher (Data Bus)



3.2.4 Der Fetch-Zyklus

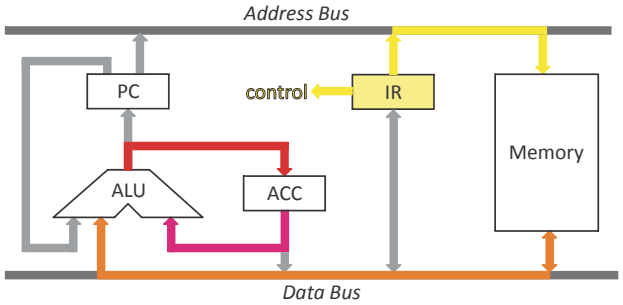
- Inhalt Programmzähler PC über den Adressbus am Speicher anlegen
- Inhalt der Speicherzelle (Befehl) über den Datenbus in das Befehlsregister (IR) einlesen
- gleichzeitig den Inhalt des PC mit der ALU um 1 erhöhen



- PC und IR sind getaktete Speicherelemente (Register);
  mit der Taktflanke wird mit dem bis zur Taktflanke gültigen PC der Befehl ausgelesen,
  nach der Taktflanke steht im PC der um 1 inkrementierte Wert, der dann für das Auslesen des Speichers nicht mehr relevant ist, weil ja die Taktflanke schon vorüber ist.

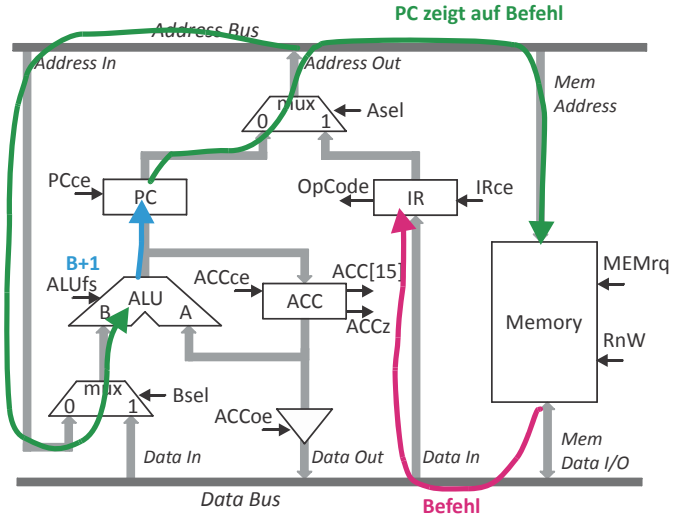
3.2.5 Decode / Execute-Zyklus

- In diesem Beispiel verarbeitet die ALU 2 Operanden (Beispiel Addition ADD):
  - o einer aus dem Akku (ACC)
  - o ein weiterer aus dem Speicher
- der Befehl, gespeichert im Befehlsregister IR, enthält Bits für die Steuerung „control“ und Bits, die die Adresse des 2. Operanden darstellen



- der Adressteil des Befehls wird über den Adressbus an den Speicher angelegt, der 2. Operand wird aus dem Speicher zur ALU geschaltet
- ALU führt Rechenoperation durch und speichert das Ergebnis im Akkumulator ACC.

- **Datenpfad beim Fetch-Zyklus LDA**
- nach der Taktflanke wird ist der Wert des PC inkrementiert



Decode/Execute-Zyklus 1.Befehl

- mit dem Maschinencode „0“ wird die Adresse „01D“ auf den Adressbus gegeben,
- der Speicher auf Ausgabe geschaltet und
- der ausgegebene Speicherinhalt in den Akku ACC eingelesen
- der Inhalt der Speicherzelle der Adresse 01D steht dann im ACC mit dem Wert „0ABC“

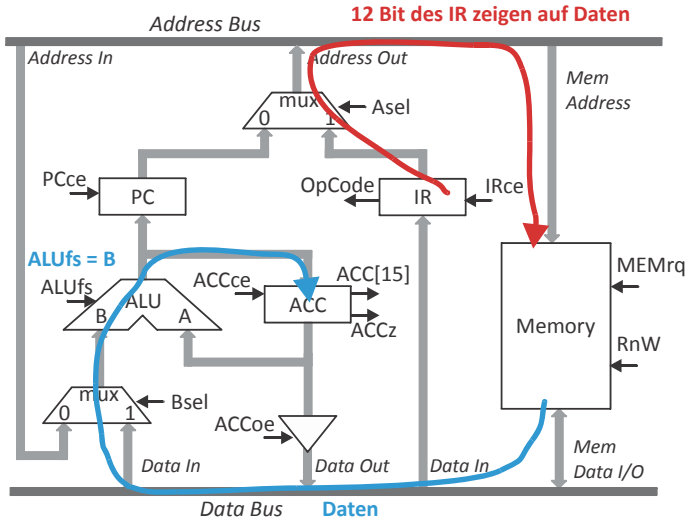
MU0

PC **1**
ACC **0ABC**
IR **001D**

Memory

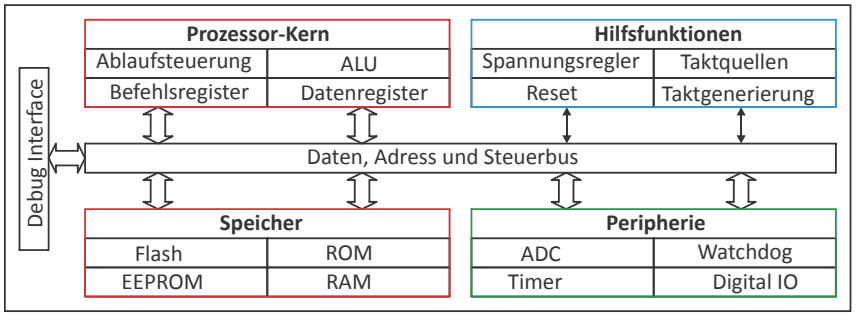
	Assembly Mnemonic	Machine Code
Program Address 000	**LDA 01D**	**0 01D**
001	ADD 01E	2 01E
002	STA 01F	1 01F
003	STP	7 000
004	---	---
005	---	---
Data 01D	ABC	
01E	111	
01F	---	

- **Datenpfad bei Decode/Execute von LDA**
- die Daten aus dem Speicher werden über die ALU zum ACC „durchgeschleift“



4 Prozessorkern und Speicher des STM8

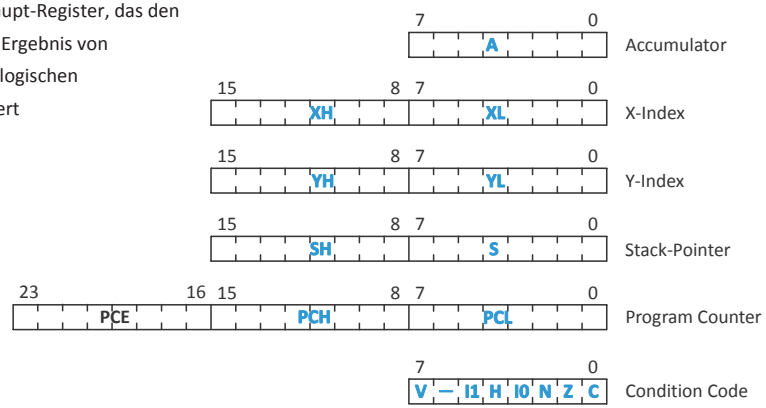
4.1 Übersicht System-on-Chip



4.2 Der Prozessor-Kern aus Sicht des Programmierers

- 8-Bit Architektur
- 16-Bit Operationen auf den Indexregistern

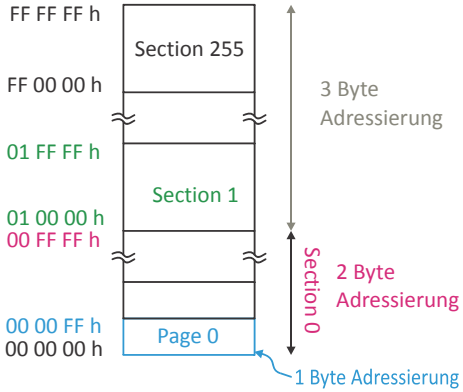
- Akkumulator (A): Haupt-Register, das den Operanden und das Ergebnis von arithmetischen und logischen Operationen speichert



4.3 Der Speicher

4.3.1 Adressierung

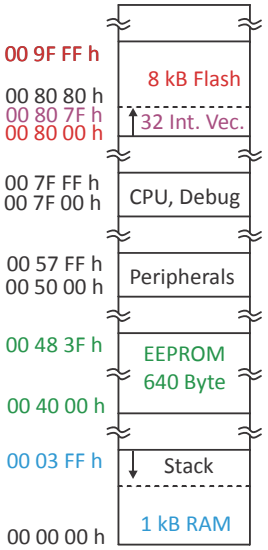
- Adressierung kann mit 1, 2 oder 3 Byte erfolgen
- Mit 1 Byte kann eine Seite ***page*** mit 256 Bytes adressiert werden,
- mit 2 Bytes eine Sektion ***section*** mit 64 kB,
- mit 3 Bytes der gesamte Bereich von 64 MB.
- Der Speicher für Programm und Daten ist linear, d.h. jede Adresse kommt nur einmal vor und an jeder Adresse des gesamten adressierbaren Bereichs von 64 MB liegt nur eine Funktion vor:
  - RAM
  - EEPROM oder Flash
  - Tabellen / Vektoren für Interrupt und Reset
  - Eingabe / Ausgabe (IO), Peripherie



4.3.2 Organisation des gesamten Speicherbereichs

- Das Diagramm rechts wird ***Memory map*** genannt: Bezeichnung *Memory map* ist eigentlich nicht ganz richtig, da neben dem
  - Speicher (Memory, bestehend aus RAM, EEPROM, Flash)
  - auch die Adressen für den Zugriff auf
    - Peripherie (I/O, Schnittstellen, Funktionen, Pins),
    - die Lage des Stacks im RAM,
    - spezielle Konfigurations-Bits im EEPROM (hier nicht gezeigt),
    - die Lage der Vektoren (Adressen) für den Reset und verschiedene Interrupts, die im Flash untergebracht sind,
    - und die Register der CPU, das Debug Interface und den Interrupt Controller (ITC)

  dargestellt sind.
- Da die Adressierung und der Zugriff auf die Peripherie wie der Zugriff auf den Speicher erfolgt, spricht man von ***Memory mapped IO***.



5 Befehlssätze und Assembler des STM8

Der Befehlssatz kann nach verschiedenen Kriterien strukturiert werden:

- Länge des Befehls im Maschinencode (1 bis 5 Byte)
- Adressierungsart (z.B. keine Angabe einer Adresse erforderlich, direkte Adresse als Bestandteil des Befehls, Adresse indirekt in einem Register)
- Gruppierung nach der Art des Befehls (z.B. Speicherbefehle, Vergleichsbefehle, logische Operationen, Verzweigungen, Interrupt Verarbeitung)
- Assemblerbefehle (OpCode) bestehen aus 3 bis 5 Buchstaben, die groß geschrieben werden, danach folgt mit einem Leerzeichen der Operand, der Adressen, Register oder Zahlenwerte darstellen kann

- Beispiel Additionsbefehl:
  - o ADD **Add** (without carry)
  - o ADC **Ad**d with **C**arry
  - o ADDW **Add w**ord without carry
- Die Assemblerbefehle werden Mnemonics genannt (vom griechischen *mnēmoniká* „Gedächtnis“) Für die Darstellung des Operanden werden auch runde Klammer, eckige Klammern, Nummern- und Dollar-Zeichen verwendet

- einige Adressierungsarten am Beispiel des Befehls ADC:

Mnemonic Operand	Maschinencode im Speicher	Funktion
o ADC A,#$64	A9 64	Addiere die Zahl 64h (=100d) und den Carry zum aktuellen Inhalt des Akku (unmittelbare Adressierung, immediate)
o ADC A,$1000	C9 10 00	Addiere den Inhalt des Speichers mit der Adresse 1000h und den Carry zum aktuellen Inhalt des Akku (direkte, absolute Adressierung); beim STM8 steht im Speicher nach dem OpCode C9 zuerst das High-Byte 10, dann das Low-Byte 00 der Adresse (*Big Endian*)
o ADC A,(X)	F9	Addiere den Inhalt der Speichers mit der Adresse, die durch den Inhalt des X-Registers bestimmt wird, und den Carry zum aktuellen Akkumulator Inhalt (Register-indirekte Adressierung)

- Wenn alle Befehle mit allen Adressierungsarten kombiniert werden können, spricht man von einem *orthogonalen* Befehlssatz;
  bei einigen modernen Prozessoren kann jeder Befehl auch mit jeder Bedingungsabfrage (Flags C, Z, V) verknüpft werden; Nachteil: Befehle werden sehr lang (z.B. 32 Bit)

5.2.2 Unmittelbar (immediate) Adressierung

Der zu verarbeitende Operand folgt nach dem OpCode. Befehle mit möglicher unmittelbarer Adressierung sind zum Beispiel: LD, MOV, CP, BCP, AND, OR, ADC, SUB

```
05BA    AE FF   LD X,#$FF      ; Lade das X-Register mit dem Wert FFh
05BC    A3 55   CP X,#$55      ; Vergleiche das X-Register mit dem Wert 55h,
                                 (X-55h), und setze die Flags N, V, Z
05BE    A6 F8   LD A,#$F8      ; Lade den Akkumulator mit F8h
```

5.2.3 Direkte Adressierung

Wird auch als absolute Adressierung bezeichnet;
die Adresse kann 1 Byte (short), 2 Byte (long) oder 3 Byte (extended) sein;
(extended werden wir nicht verwenden müssen, da der gesamte Adressraum unseres Mikrocontrollers in der untersten Sektion bis 64 kByte liegt);
bei den meisten Assemblerbefehle ist eine direkte Adressierung mit 1 oder 2 Byte-Adressen möglich.
Beispiel: Die Adresse sei ein Wort und erlaube daher einen Adressbereich von 0000 bis FFFF, was 2 Bytes für die Adresse nach dem OpCode erfordert:


```
0409  C6 06 E5      LD A,$06E5
06E5  40            coeff dc.b $40
```

Aktion:

```
A = (coeff)
  = ($06E5)
  = $40
```