Rechnerstrukturen und Maschinennahe Programmierung

Prof. Dr.-Ing. Tim Tiedemann
basierend auf Folien von Prof. Dr.-Ing. Andreas Meisel

```
                ldr   r0,=VariableA
                ldrh  r1,[r0]
                ldr   r2,[r0]
                str   r2,[r0,#VariableC-VariableA]

@ Zugriff auf Felder (Speicherzellen)

                ldr   r0,=MeinHalbwortFeld
                ldrh  r1,[r0]
                ldrh  r2,[r0,#2]
                mov   r3,#10
                ldrh  r4,[r0,r3]

                ldrh  r5,[r0,#2]!
                      0,#2]!
                      0,#2]!

@ Adc                 von ur
                      inWort
                      ]
                ldr   r2,[r0,#4]
                adds  r3,r1,r2
```

```
double GetMeasurementV
   unsigned char ADC_S
   unsigned int  ADC_V
   unsigned int  ADC_V
   double        Volta

    /*  AD-Wandlung wi                    lNr in ADC CONTROL gestartet */
    out8(ADC_CONTROL,

    /* Auf das Ende de                    ling) */
    do{
       ADC_Status = in8(ADC_REG_STATUS);
    }while((ADC_Status & 0x80) == 0);
```

```
                _LOW);
                _HIGH);

                lueHigh &          Bits ausblend. */
                High << 8          menführen      */

                MAX_VOLTA
```

Agenda – Kapitel 1

0. - Formalien

   - Was heißt „maschinennahes Programmieren“?

1. Motivation, Grundlagen des „maschinennahen Programmierens“
2. Grundsätzliches zum Aufbau von Rechnern
3. Daten
4. Befehle und Befehlscodierung
5. Maschinennahes Rechnen

Vorlesung

- Vortragsfolien → (nach/vor der Vorlesung in EMIL)
- Vertiefungen → Tafel, (nach Vorlesung in EMIL)
- Übungsaufgaben → Tafel, (nach Vorlesung in EMIL)
- Programmierbeispiele → Tafel, (nach Vorlesung in EMIL)

Besondere Daten

05.10.2017 keine RMP-Veranstaltung! (Dienstagsveranstaltungen → Donnerstag)
07.12.2017 keine RMP-Veranstaltung! (Konferenz)
18.01.2018 Klausurvorbereitung

Alle Materialien (zu Vorlesung und Praktikum):
→ EMIL-Raum „Rechnerstrukturen und Maschinennahe Programmierung (Tdm) W17“

Klausur

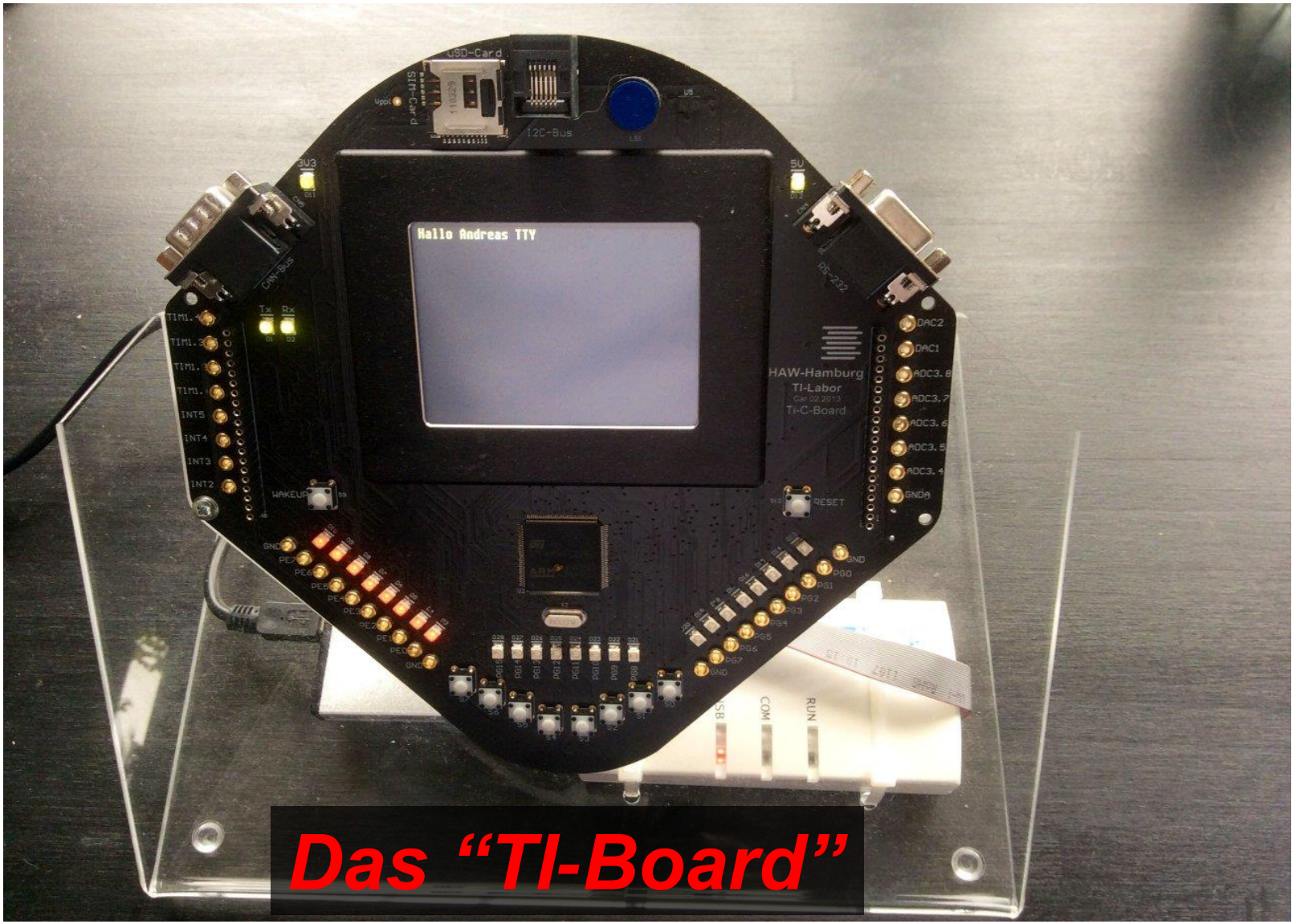
- Teilnahmevoraussetzung (PVL) : erfolgreich bestandenes Praktikum, s.u.
- benotet

Praktikum

- Anwesenheitspflicht (Prüfungsvorleistung)
- Sie müssen angemeldet sein (sollte bereits erfolgt sein!)
- Gruppenarbeit: je **2** Personen arbeiten zusammen
- 4 Versuche (Assembler, C, IO-Programmierung, ….)
- Versuch thematisch vorbereiten (Vorbereitungsaufgaben in Aufgabenblättern)

  → **Test** zu Beginn des Praktikums (15 Min.) → pünktlich erscheinen

- 50% der Punkte aller Tests müssen erreicht werden (PVL)
- Versuche müssen erfolgreich abgeschlossen (abgenommen) werden (PVL)


uSD-Card
I2C-Bus
Hallo Andreas TTY
CAN-Bus
RS-232
HAW-Hamburg
TI-Labor
Ti-C-Board
RESET
WAKEUP
COM
RUN

Das “TI-Board”

Was heißt „maschinennahes Programmieren“?

Hochsprachenebene:
- Rechner wird als abstrakte Maschine aufgefasst.
- Die konkrete Arbeitsweise ist uninteressant.

→ Java, C/C++, Fortran,...

Maschinennahe Ebene (Registerniveau):
- Verstehen des Rechners auf Registerniveau
- typisch: technische Informatiker, Applikationsingenieure

→ Assembler, C/C++

Gatter- und Schaltwerkniveau:
- Verstehen der Prozessorkomponenten
- typisch: Hardwareentwickler

→ Hardware-Beschreibungssprachen, Schaltpläne

Transistorniveau:
- Verstehen der integrierten Schaltkreise (
- typisch: theoretische Elektrotechniker, Physiker

→ Schaltpläne, Layout-Entwurf

1. Grundlagen des „maschinennahen Programmierens“

1.1 Inhalt und Aufbau der Vorlesung

1.1.1 Grundlegende Funktionsweise von Computern

Inhalte:

- einfaches Computermodell :
  Speicher, CPU, ALU, Steuereinheit, Busse, IO-Einheiten
- Datencodierung und Befehlscodierung auf Maschinenebene
- Grundbefehlssatz von Computern
- Adressierungsarten für den Datenzugriff
- (neu, gegen Ende: Eigenschaften aktueller PC-Hardware)

vermittelte Fähigkeiten:

- *Maschinenprogramme und Speicherdumps lesen können*
- *maschinennahe Operationen lesen und programmieren können*

1.1.2 Assemblerprogrammierung am Beispiel des Cortex-M4-Prozessors

Inhalte

- Register
- Basisadressierungsarten
- elementarer Befehlssatz
- Unterprogrammtechniken auf Maschinenebene
- Ein-/Ausgabeoperationen

vermittelte Fähigkeiten

- *Assembler-Programme programmieren und debuggen können*
- *Assembler-Programme strukturiert aufbauen können*
- *Unterprogrammtechniken auf Maschinenebene einsetzen können*

1.1.3 Grundkonzepte der maschinennahen Computerprogrammierung

Inhalte:

- notwendige Fähigkeiten maschinennaher Hochsprachen
- Strukturierung, Typen und Abstraktion
- Kennzeichen prozeduraler Programmiersprachen

vermittelte Fähigkeiten:

- *Algorithmen strukturiert umsetzen können*
- *Daten strukturieren können*

1.1.4 Programmiersprache C

Inhalte:

- grundlegende Sprachkonstrukte von C
- Zeiger und Zeigerarithmetik
- Parameterübergabe bei C-Funktionen
- Modularisierung von C-Programmen
- Datenstrukturierung in C-Programmen
- Maschinennahe Programmierung mit C

vermittelte Fähigkeiten:

- *C-Programme schreiben und debuggen können*
- *C-Programme strukturieren und modularisieren können*
- *Zeiger verstehen*
- *IO-Einheiten programmieren können*

1.2 Anwendungsgebiete der maschinennahen Programmierung

Hardware-Einbindung / Gerätetreiber

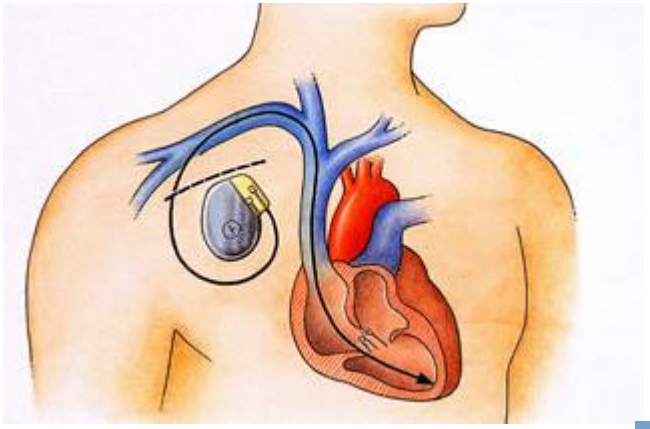
- Digitale/Analoge Schnittstellen (Meßsignalaufnehmer, Audiointerface)
- Kommunikationsschnittstellen (USB, COM, Netzwerkkarten,...)
- Mediainterfaces (Soundkarten, Joysick, MIDI, Video, ...)

Embedded Controller (Prozessor u. E/A-Schnittst. auf einem Chip für wenige Euro)

- Haushaltsgeräte (Schaltuhren, Fernbedienungen, Waschmaschinen, ...)
- Spielzeuge (Playstations, LEGO-Mindstorm, ...)
- Autos, Digitalkameras, Smartphones, Reader, Navigationssysteme, ...
- Maschinensteuerungen
- Mit Netzzugang: „IoT Device“ („Internet of Things“)

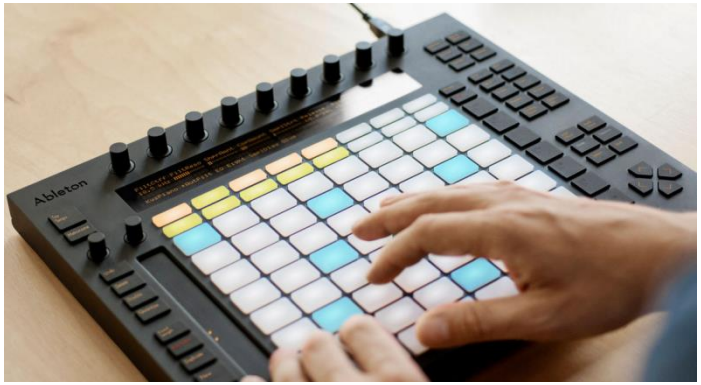
Sicherheitsrelevante Programme

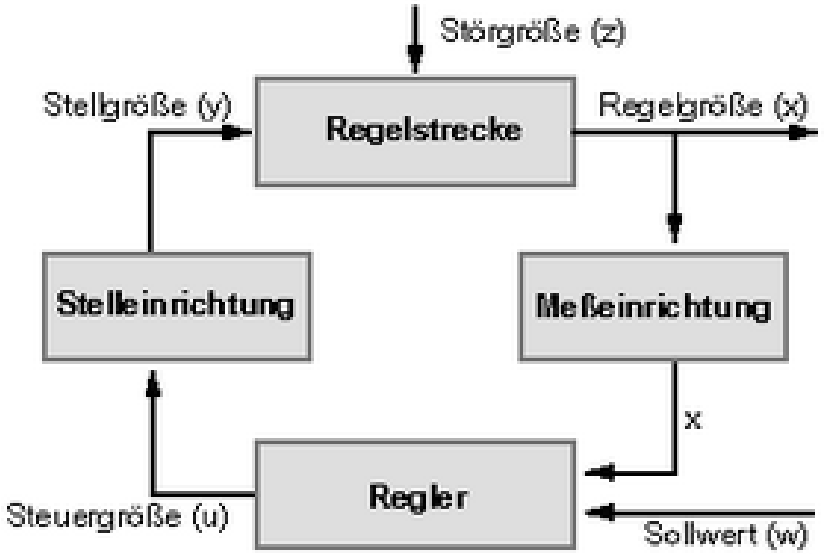
- Medizintechnik (z.B. Herzschrittmacher, Herz-Lungen-Maschine)
- Kraftwerkstechnik, Flugzeugtechnik
- Bremsregelung, ABS



Geschwindigkeitskritische Anwendungen

- Audio- und Videoanwendungen, Musikanwendungen
- Echtzeitsysteme und Regelungen

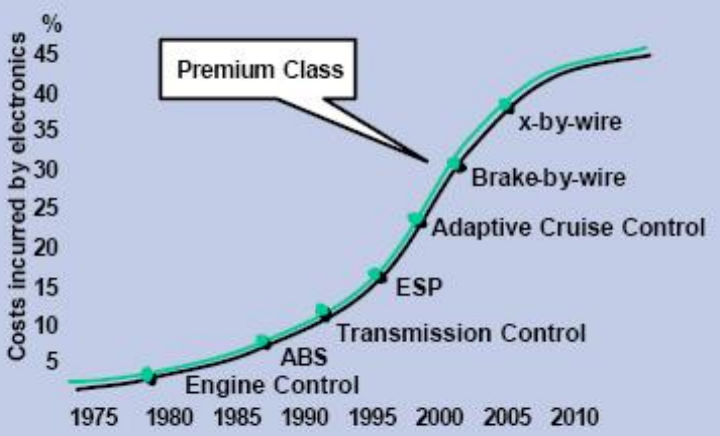




Entwicklunge eingebetteter Systeme in Fahrzeugen

Electronic Content

The continuous growth of vehicle electronics leads to a significant increase in software complexity



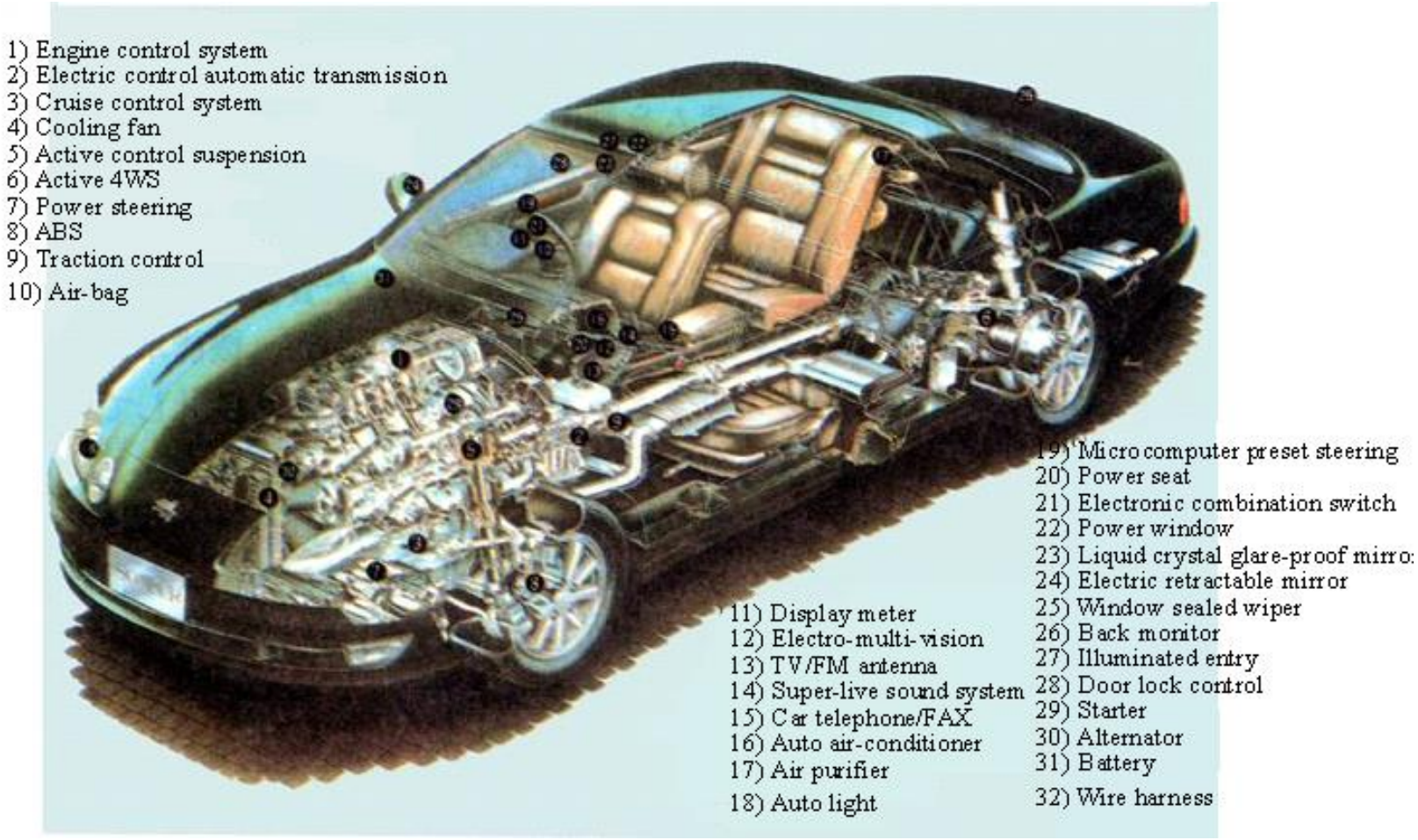
- more than 80% of functions driven by software
- continuously increasing

Premium Class, 2000

State of the Art

... not even accounting for telematics

Premium Class, 2000



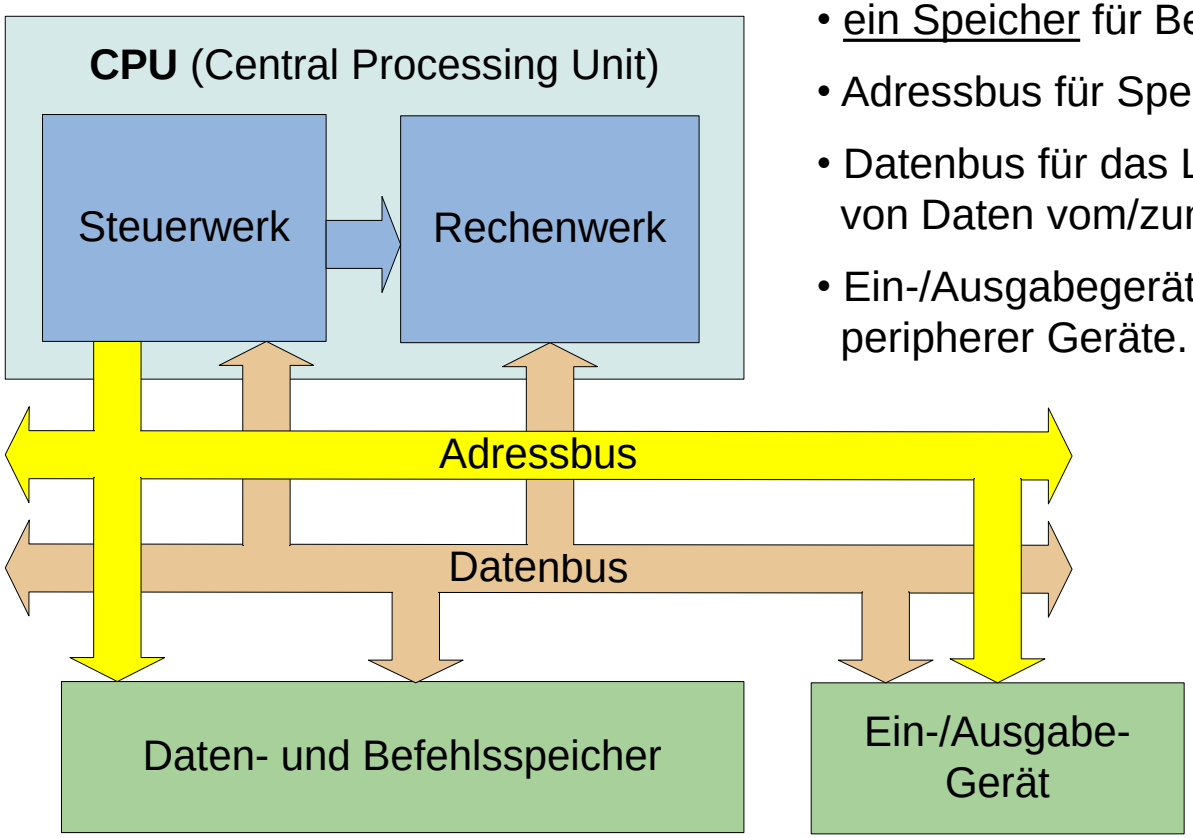
Provided by Toyota Motor Corporation

2. Grundsätzliches zum Aufbau von Rechnern

2.1 Von-Neumann-Rechnermodell

2.1.1 Übersicht

Von-Neumann-Rechnermodell



- ein Speicher für Befehle und Daten
- Adressbus für Speicheradressierung
- Datenbus für das Lesen/Schreiben von Daten vom/zum Speicher
- Ein-/Ausgabegerät zur Anbindung peripherer Geräte.

2.1.2 Daten- und Befehlsspeicher

2.1.2.1 Bit = kleinstmögliche Datenmenge

1 Bit =

- ein Schaltelement (*Flip-Flop*), welches 2 Zustände einnehmen kann (0/1),
- stellt die kleinstmögliche *Datenmenge* dar,
- kann genau eine Ja/Nein-Entscheidung speichern.

n Bit können 2^n Zustände speichern:

1 Bit	0/1	***0***	***1***					
2 Bit	0/1 0/1	***00***	***01***	***10***	***11***			
3 Bit	0/1 0/1 0/1	***000***	***001***	***010***	***011***	***100***	***101***	***110***	***111***

Fragen: Speicherung von Daten und Befehlen

Angenommen ein Zeichencode umfasst 80 unterschiedliche Zeichen.
(26 Kleinbuchstaben, 26 Großbuchstaben, 10 Zahlzeichen, 18 Sonderzeichen)

Wieviele Bits sind zur Codierung eines Zeichens mindestens notwendig?

Wie viele Zustände lassen sich mit 12 Bit codieren?

Wieviele Bits sind notwendig, um die 256 Graustufen eines Bildpunktes unterscheiden zu können?

2.1.2.2 Byte = kleinste Zugriffseinheit

1 Byte =

- eine Gruppe aus 8 Bits,
- kann 256 unterschiedliche Zustände einnehmen,
- Ist die kleinste schreib- und lesbare Datenmenge in den meisten Computern (→ *Bytemaschinen*).

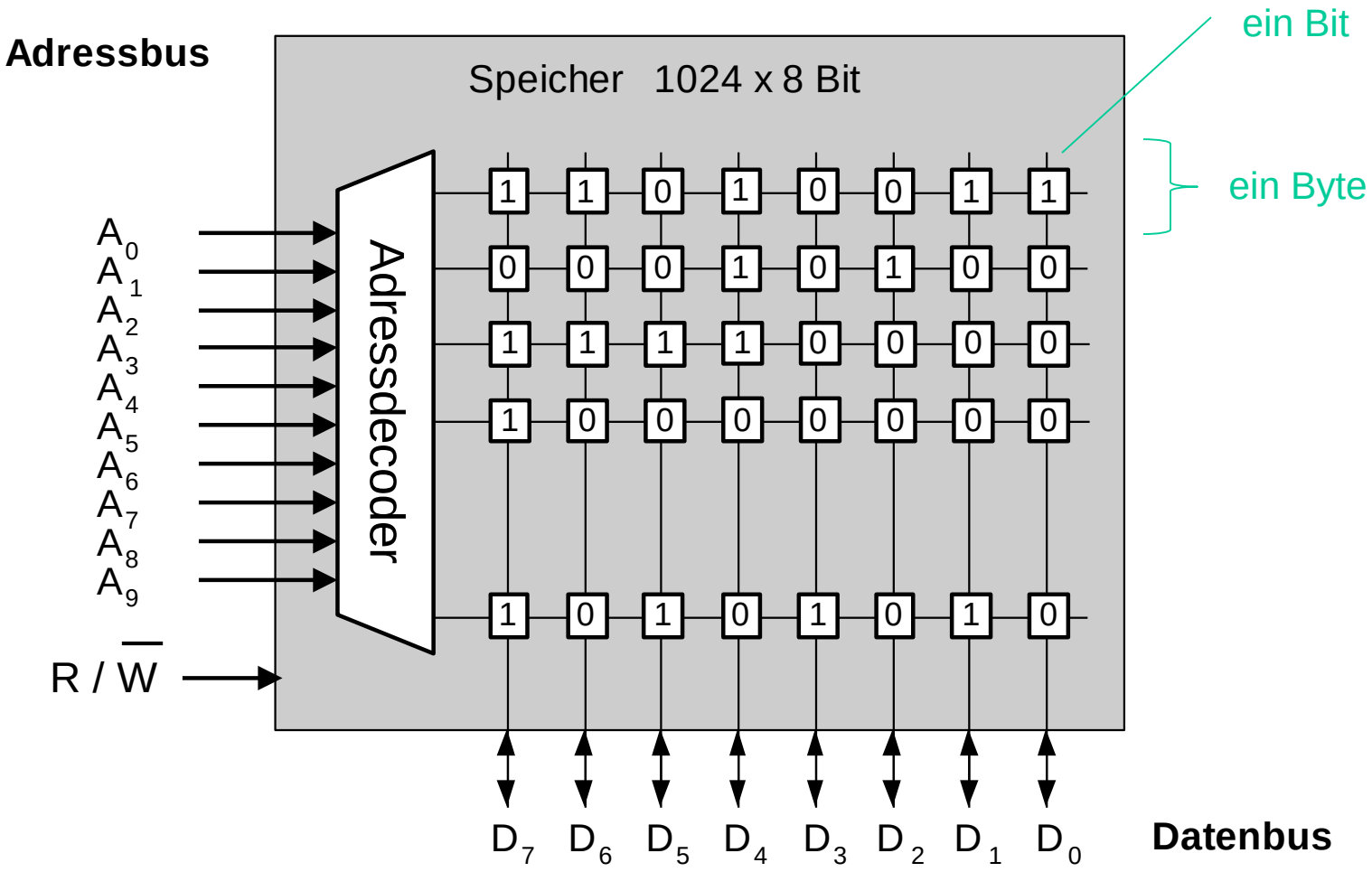
- Jedes Byte eines Speichers ist durch eine eigene Adresse ansprechbar (in Bytemaschinen).

Daten bestehen meist aus einer ganzzahligen Anzahl von Bytes.

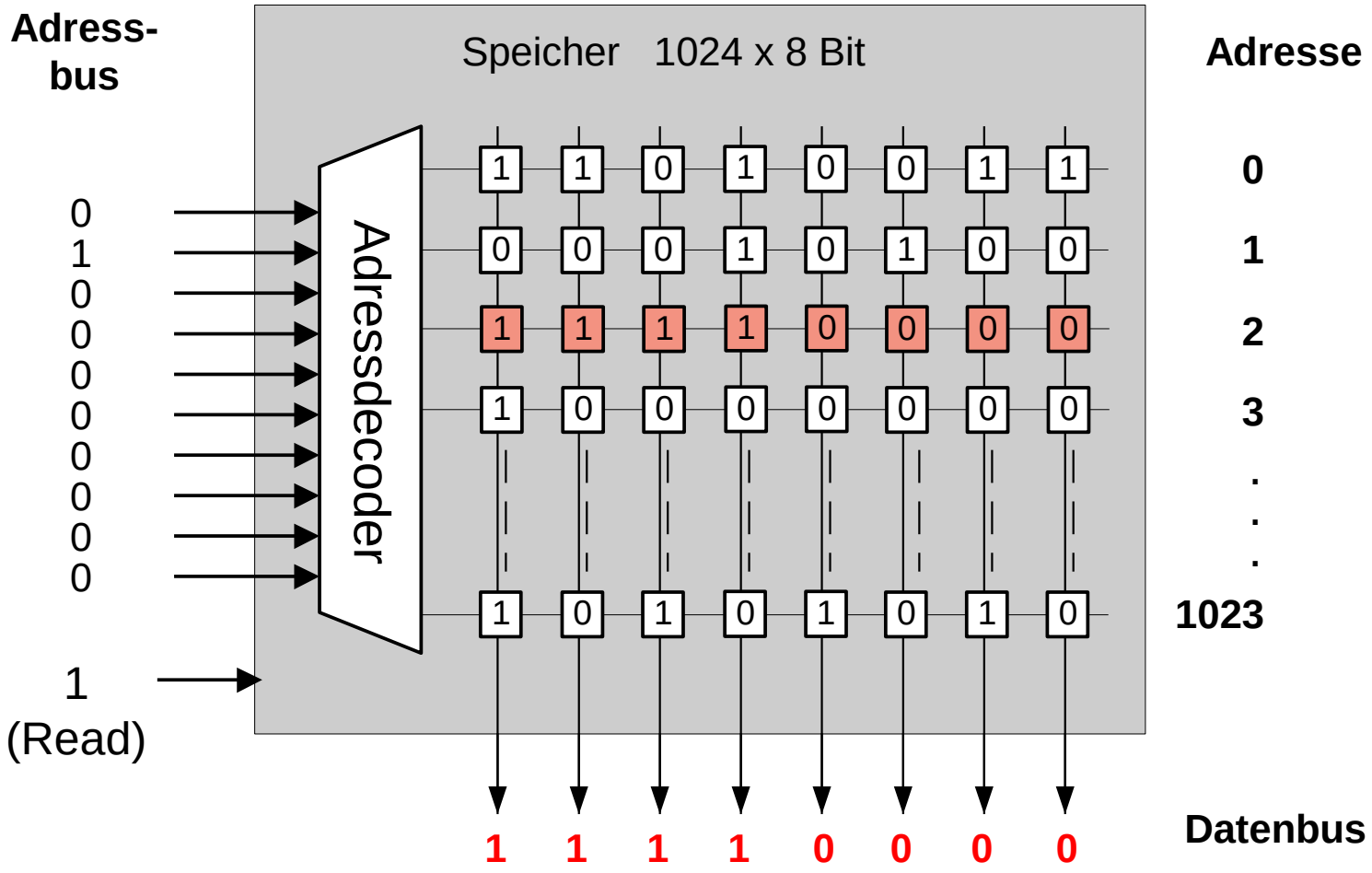
Beispiele:	Musiksignale	→ 2, 3 oder 4 Byte
	Farbcodierung eines Farbbildpunktes	→ 3 Byte

2.1.2.3 Realisierung des Daten- und Befehlsspeichers (einf. Beispiel)

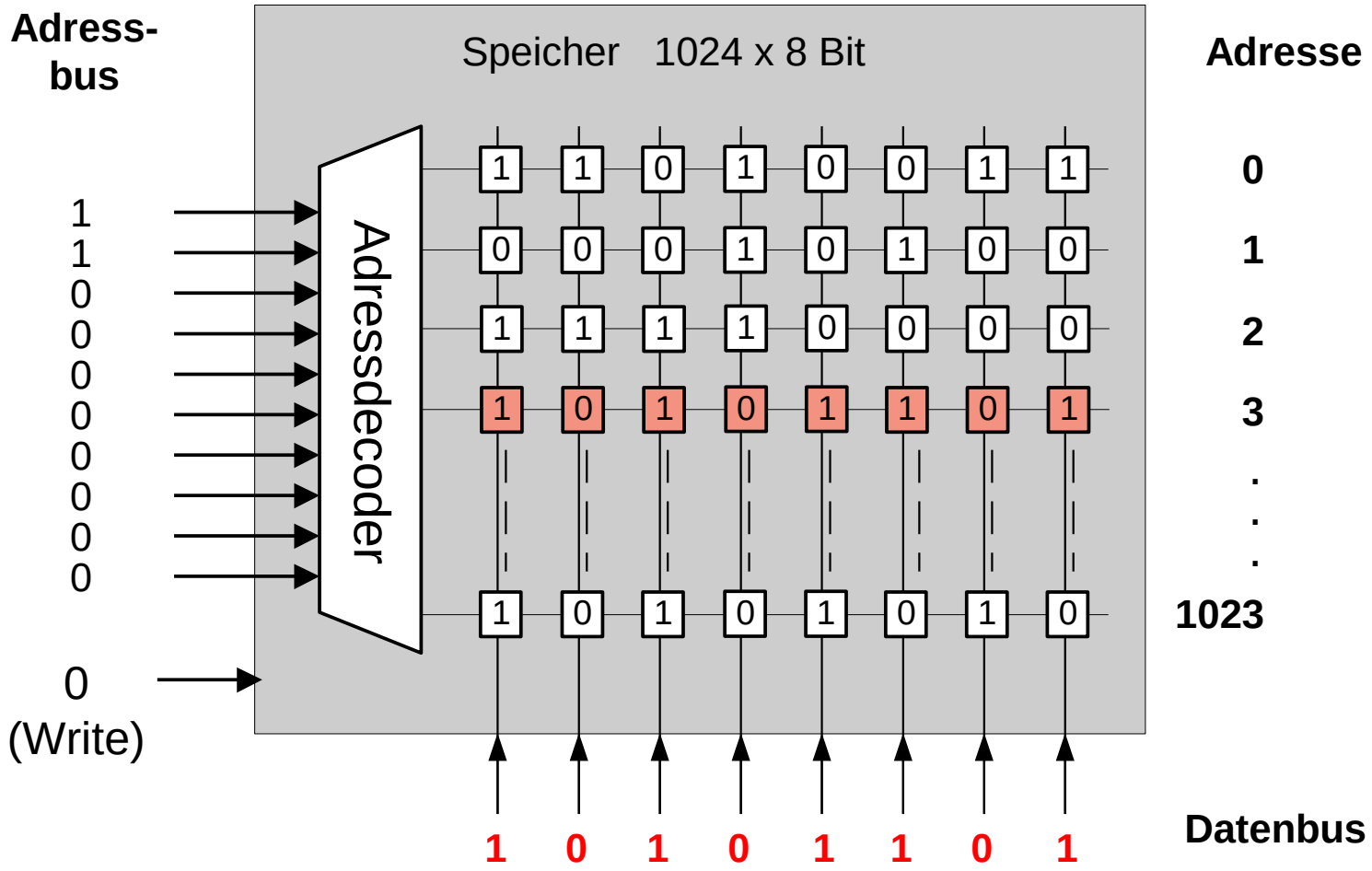
Zweck: Speichern und Lesen von Daten und Maschinenbefehlen unter einer Adresse (hier Bytemaschine).



Lesen eines Speicherwortes



Schreiben eines Speicherwortes



2.1.3 Adress-/Datenbus verschiedener Rechner

- typ. Adressbusbreiten:
  - 16 bit (z.B. 6502) → Adressraum 2^{16} = 65.536
  - 24 bit (z.B. 68000) → Adressraum 2^{24} = 16.777.216
  - 32 bit (**Cortex M4**) → Adressraum 2^{32} = 4.294.967.296
  - 36 bit (Pentium II, .. IV) → Adressraum 2^{36} = 68.719.476.735

- typ. Datenbusbreiten:
  - 8 bit (z.B. 6502)
  - 16 bit (z.B. 68000)
  - 32 bit (z.B. 68030, **Cortex M4**)
  - 64 bit (z.B. Pentium)

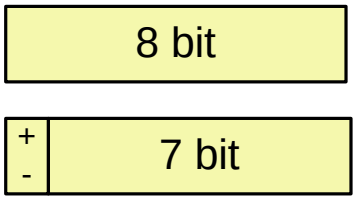
- übliche kleinste adressierbare Einheit ist **1 Byte** → *Byte-Maschine*

2.1.4 Maschinennahe Datenformate (Anm.: Bezeichnungen maschinenabh.)

Byte-Format (1 Byte)

unsigned 0 ... 255 (2^8-1)

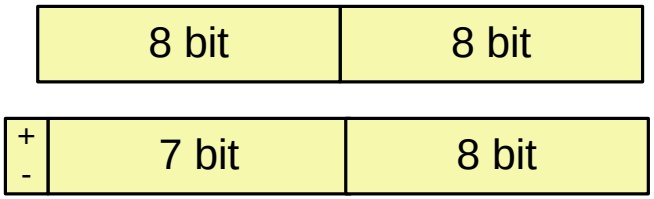
signed -128 ... 127



Halbwort-Format (2 Byte)

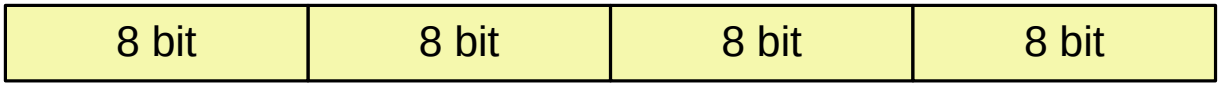
unsigned 0 ... 65 535 (2^{16}-1)

signed -32 768 ... 32 767

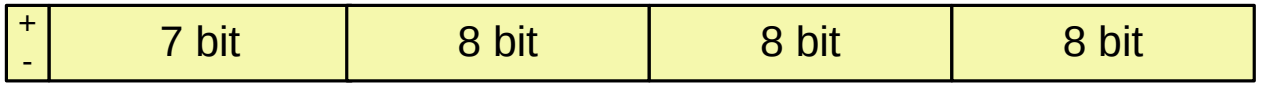


Wort-Format (4 Byte)

unsigned 0 ... 4 294 967 295 (2^{32}-1)



signed -2 147 483 648 ... 2 147 483 647



Hinweis zur Darstellung von Speicherauszügen (Dumps)

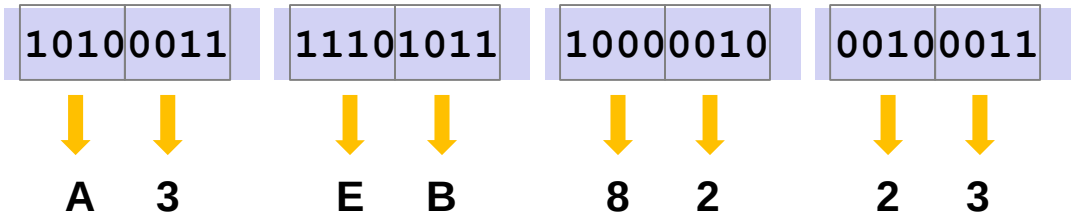
Eine größere Bitgruppe ist für den Menschen kaum lesbar/merkbar.

Beispiel : 4 Byte

10100011	11101011	10000010	00100011

Eine kompaktere Darstellung erhält man wie folgt:

1. Jedes Byte wird in zwei Hälften (*Nibble* = 4 bit) unterteilt.
2. Der Wert eines Nibbles wird dann kurz durch seinen Hexadezimalwert dargestellt.



Bit-muster	Hex-wert
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Fragen: Darstellung von Speicherauszügen (Hexdump)

Gegeben ist der folgende Hexdump einer Bytemaschine ab Adresse 1000:

```
A4 33 72 F9 B0
```

1. Wie viele Bits und Bytes sind dargestellt ?
2. Was ist die Adresse des letzten Bytes?
3. Geben Sie das gespeicherte Bitmuster an.
4. Handelt es sich um codierte Buchstaben, Zahlen oder Befehle?

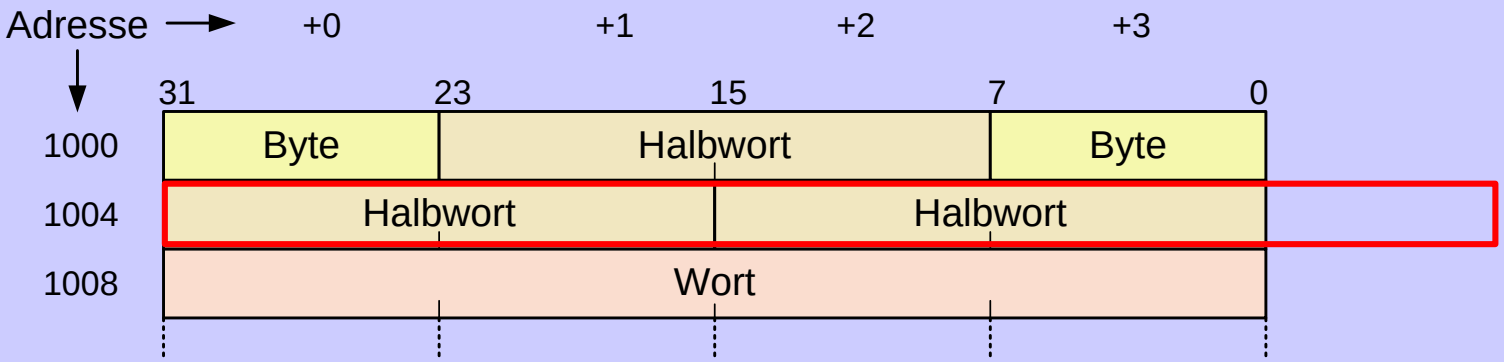
2.1.5 Speicherorganisation

Die Daten können auf unterschiedliche Art im Speicher abgelegt werden:

2.1.5.1 n-Bit-Speicher (= n-Bit-Zugriffsbreite = n-Bit-Datenbusbreite)

→ d.h. es können n Bit in einem Zugriff gelesen/geschrieben werden
typ.: 16-Bit-Speicher, 32-bit-Speicher, 64-bit-Speicher

Beispiel: **32-bit-Speicher** ab Adresse 1000



Anm.: Nicht immer braucht man 32 bit (z.B. Bildpunkte, ASCII-Zeichen, ...).
Um keinen Speicher zu verschwenden, müssen auch 16 bit oder 8 Bit adressierbar sein. → deswegen „Byte-Maschine“

*2.1.5.2 Byte-Reihenfolge (byte ordering) bei **Halbworten** (2 Byte) u. **Worten** (4 Byte)*

big endian: „*höchstwertiges*“ Byte auf niedrigster Adresse

little Endian: „*höchstwertiges*“ Byte auf höchster Adresse (sog. *byte swapping*)

Beispiel: Folgende Daten sind im Speicher ab Adresse 1000 abgelegt (Hex.):

Word:	(4-Byte)	`12 34 56 AB`
Halfword:	(2-Byte)	`41 44`
Byte:		`78`
Byte:		`89`
String:		`„ABCD“`

big-endian

Adr.	+0	+1	+2	+3
1000	**12**	**34**	**56**	**AB**
1004	**41**	**44**	**78**	**89**
1008	**41**	**42**	**43**	**44**

little-endian

byte swapping (nur) bei Worten und Halbworten im Speicher

Adr.	+0	+1	+2	+3
1000	**AB**	**56**	**34**	**12**
1004	**44**	**41**	**78**	**89**
1008	**41**	**42**	**43**	**44**

2.1.5.3 Speicherausrichtung (data alignment)

aligned: Worte und Langworte werden so abgelegt, dass sie in einem Zugriff gelesen werden können.

→ Worte beginnen auf Adr., die durch 4 teilbar sind (Wortgrenzen).

→ Halbworte beginnen auf Adr., die durch 2 teilbar sind (Halbwortgrenzen).

Beispiel: 32-Bit-Zugriffsbreite, big-endian,
folgende Daten sind im Speicher abgelegt (Hex.):

Halfword:	(2-Byte)	12 34
Word:	(4-Byte)	56 78 9A BC
Byte:		CD CE CF
Halfword:		1F A1
Halfword:		B5 D5

aligned

Adr.	+0	+1	+2	+3
1000	12	34	xx	xx
1004	56	78	9A	BC
1008	CD	CE	CF	xx
	1F	A1	B5	D5

non-aligned

Adr.	+0	+1	+2	+3
1000	12	34	56	78
1004	9A	BC	CD	CE
1008	CF	1F	A1	B5
	D5			

BEISPIEL: Cortex M4 im Praktikumssystem

Alignment:	Worte	immer **a**ligned an Wortgrenzen
	Halbworte	immer **a**ligned an Halbwortgrenzen
Byte-ordering:	**l**ittle **e**ndian (d.h. byte swapping bei Halbworten und Worten)	

Folgende Daten sind aligned im Speicher abgelegt
(Anm.: Hexadezimaldarstellung):

Halfword:	`12 34`	→ alignment erzwingen (ALIGN 4)
Word:	`56 78 9A BC`	
Byte:	`CD CE CF`	→ alignment erzwingen (ALIGN 2)
Halfword:	`1F A1`	

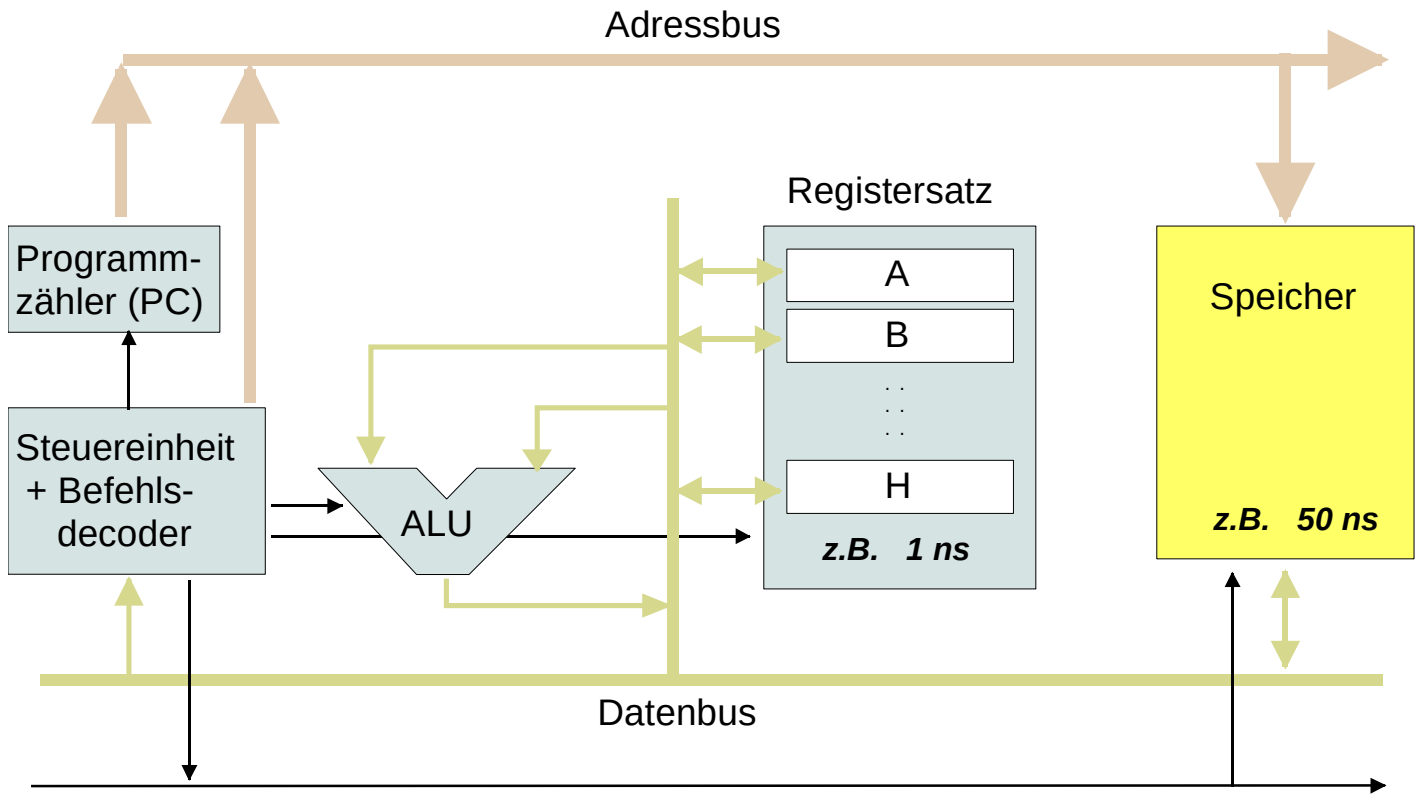
Im Debugger sieht der Speicherinhalt wie folgt aus:

`34 12 00 00  BC 9A 78 56  CD CE CF 00  A1 1F`

Würde man beim Ablegen der Daten auf das Alignment verzichten (was ohne weiteres geht), würden die Daten beim späteren Lesezugriff falsch gelesen.

2.1.6 Zentraleinheit (CPU = central processing unit)

….. eines einfachen Von-Neumann-Rechners
→ gemeinsamer Speicher für Befehle und Daten



Zentraleinheit (CPU)

Programmzähler : (PC = Program counter)

Enthält die Adresse des nächsten auszuführenden Befehls.

Steuereinheit bzw. Befehlsdecoder : (CU = Control Unit)

Dekodiert den Instruktionsteil (Instr.) uns steuert abhängig davon das Zusammenspiel der anderen Einheiten (Fetch- und Execute Phase).

Rechenwerk : (ALU = Arithmetic and logic unit)

Führt Berechnungen in Byte-, Word- oder Longword-Format durch z.B. ADD, ADC, SUB, bitweises AND/OR, ...

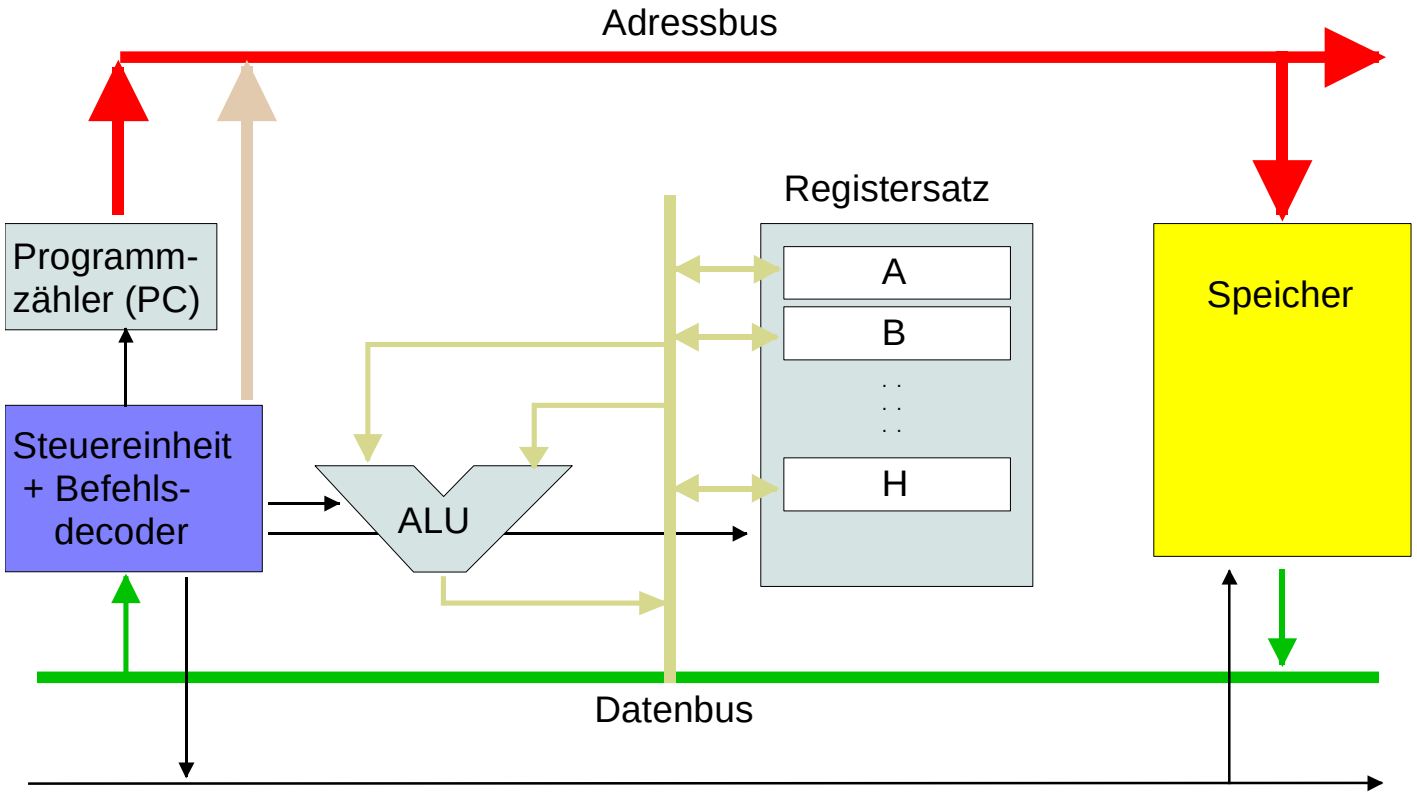
Statusregister : (CCR = Condition code register)

Mehrere 1-bit-Flags, die der Prozessor bei der Befehlsausführung setzt (z.B. Carry-Flag, Zero-Flag, Overflow-Flag, ...).

2.1.7 Befehlsausführung - "Fetch-Decode-Cycle" (Feststellen, was zu tun ist)

Beispiel: "*Lade Inhalt von Speicher 1000 in Register A*"

1. Der Programmzähler legt die aktuelle Befehlsadresse an den Adressbus.
2. Der Befehlsdecoder analysiert das Datenwort (Maschinenbefehl), um welche Instruktion es sich handelt. --> ab jetzt ist bekannt: "Inhalt von Speicher 1000 in Register A kopieren"



Befehlsausführung - ***"Execute-Cycle"*** (den Befehl Ausführen)

Beispiel: "*Lade Inhalt von Speicher 1000 in Register A*"

3. Steuereinheit legt Adresse 1000 an den Adressbus.
4. Das ausgegebene Datenwort wird nach Register A kopiert.