Rechnerstrukturen und Betriebssysteme

64-040 Vorlesung RSB
Kapitel 11: Rechnerarchitektur (I)

Norman Hendrich

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Technische Aspekte Multimodaler Systeme

Wintersemester 2025/2026

Gliederung

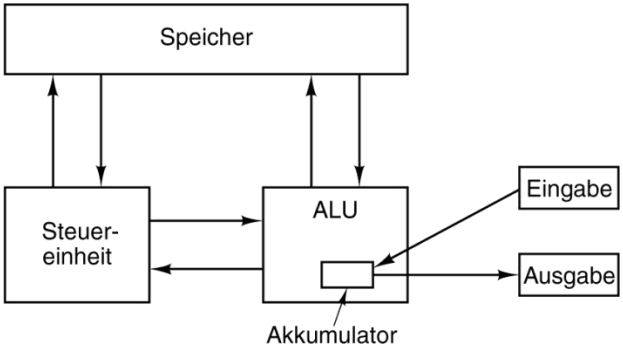
Wdh. von-Neumann Konzept

- J. Mauchly, J.P. Eckert, J. von-Neumann 1945
- Abstrakte Maschine mit minimalem Hardwareaufwand
  - System mit Prozessor, Speicher, Peripheriegeräten
  - die Struktur ist unabhängig von dem Problem, das Problem wird durch austauschbaren Speicherinhalt (Programm) beschrieben
- gemeinsamer Speicher für Programme und Daten
  - fortlaufend adressiert
  - Programme können wie Daten manipuliert werden
  - Daten können als Programm ausgeführt werden
- Befehlszyklus: Befehl holen, decodieren, ausführen

$\Rightarrow$ enorm flexibel

- **alle** aktuellen Rechner basieren auf diesem Prinzip
- aber vielfältige Architekturvarianten, Befehlssätze usw.

Wdh. von-Neumann Rechner



[TA14]

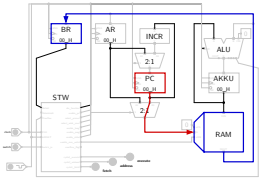
Fünf zentrale Komponenten:

- Prozessor mit **Steuerwerk** und **Rechenwerk** (ALU, Register)
- **Speicher**, gemeinsam genutzt für Programme und Daten
- **Eingabe**- und **Ausgabewerke**
- verbunden durch Bussystem

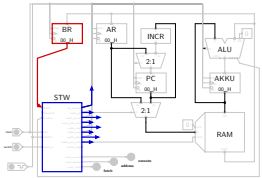
Wdh. Hardwareaufbau / Rechnerarchitektur

- Verschaltung der Hardwarekomponenten für alle mögl. Datentransfers
- abhängig vom Befehl werden nur bestimmte Pfade aktiv
- Ausführungszyklus

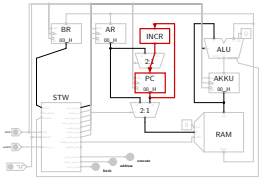
Befehl holen

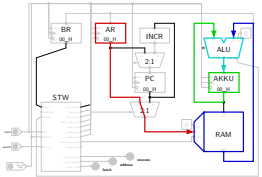


decodieren

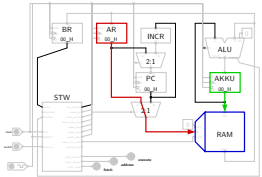


PC inkrementieren

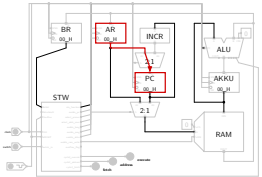




rechnen



speichern



springen

Rechnerarchitektur: Hardwarestruktur

- bis jetzt
  - Gatter und Schaltnetze
  - Flipflops als einzelne Speicherglieder
  - Schaltwerke zur Ablaufsteuerung

- weitere Komponenten: Register-Transfer- und Hauptblockebene
  - Speicher
  - Busse, Bustiming
  - Mikroprogrammierung zur Ablaufsteuerung

Speicher

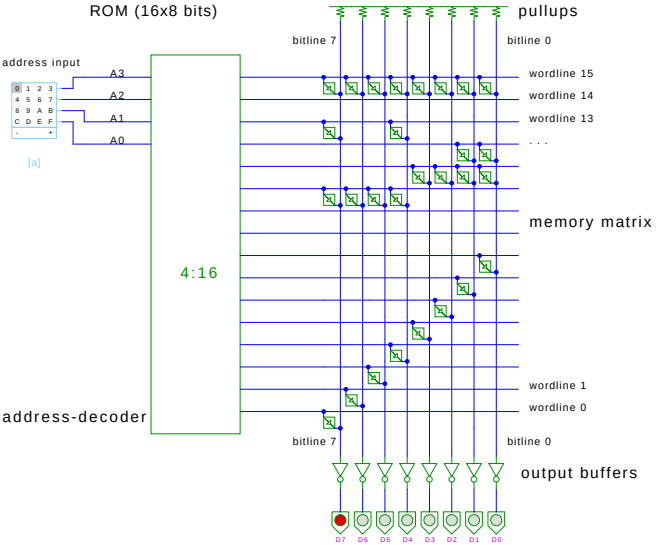
- System zur Speicherung von Information
- als Feld von N Adressen mit je m-bit Speicherworten
- typischerweise mit n-bit Adressen und $N = 2^n$
- Kapazität also $2^n \cdot m$ Bits

- Klassifikation
  - Speicherkapazität?
  - Schreibzugriffe möglich?
  - Schreibzugriffe auf einzelne Bits/Bytes oder nur Blöcke?
  - Information flüchtig oder dauerhaft gespeichert?
  - Zugriffszeiten beim Lesen und Schreiben
  - Technologie

Speicherbausteine: Varianten

Typ	Kategorie	Löschen	byte-adressierbar	flüchtig	Typische Anwendung
SRAM	Lesen/Schreiben	elektrisch	ja	ja	Cache Speicher
DRAM	Lesen/Schreiben	elektrisch	ja	ja	Hauptspeicher (alt)
SDRAM	Lesen/Schreiben	elektrisch	ja	ja	Hauptspeicher
ROM	nur Lesen	—	nein	nein	Embedded (große Stückzahlen)
PROM	nur Lesen	—	nein	nein	Embedded (kleine Stückzahlen)
EPROM	vorw. Lesen	UV-Licht	nein	nein	Prototypen
EEPROM	vorw. Lesen	elektrisch	ja	nein	Prototypen
Flash	Lesen/Schreiben	elektrisch	nein	nein	Speicherkarten, SSDs, Mobile Geräte

ROM: Read-Only Memory



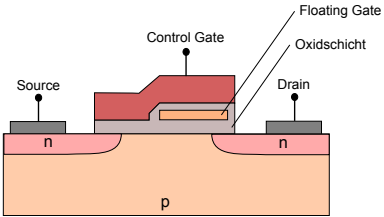
16×8 bit
4-bit Adresse
8-bit Datenwort

[Hen] Hades Demo: `40-memories/20-rom/rom`

Flash

- sehr kompakt, ein Transistor pro Zelle
- nichtflüchtig (*non-volatile*): Information bleibt beim Ausschalten erhalten

- spezielle *floating-gate* Transistoren
  - das *floating-gate* ist komplett nach außen isoliert
  - einmal gespeicherte Elektronen sitzen dort fest



- Auslesen beliebig oft möglich, schnell
- Schreibzugriffe problematisch
  - intern hohe Spannung erforderlich (Gate-Isolierung überwinden)
  - Schreibzugriffe einer „0“ nur blockweise
  - pro Zelle nur einige 10 000 . . . 100 M Schreibzugriffe möglich

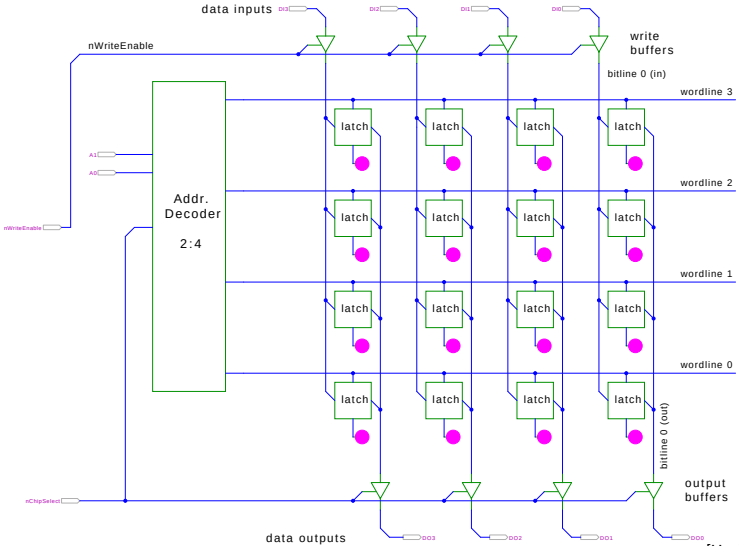
RAM: Random-Access Memory

Speicher, der im Betrieb gelesen und geschrieben werden kann

- Arbeitsspeicher des Rechners
- für Programme und Daten
- keine Abnutzungseffekte
- benötigt Spannungsversorgung zum Speichern

- Aufbau als Matrixstruktur
- n Adressbits, konzeptionell 2^n Wortleitungen
- m Bits pro Wort
- Realisierung der einzelnen Speicherstellen?
  - statisches RAM: 6-Transistor Zelle $\Rightarrow$ SRAM
  - dynamisches RAM: 1-Transistor Zelle $\Rightarrow$ DRAM

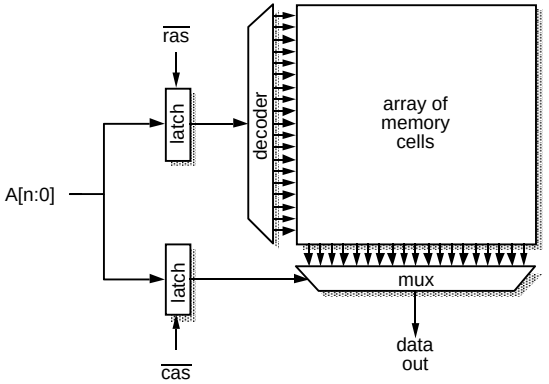
RAM: Blockschaltbild



4×4 bit
2-bit Adresse
4-bit Datenwort

[Hen] Hades Demo: `40-memories/40-ram/ram`

RAM: RAS/CAS-Adressdecodierung



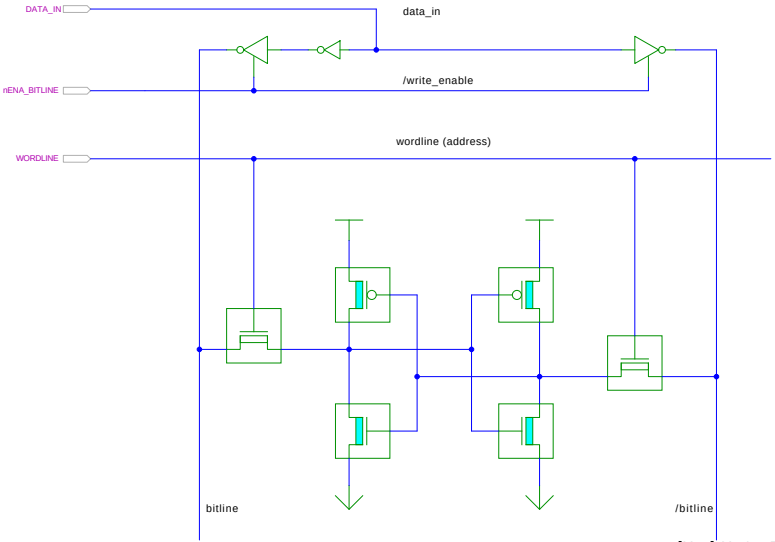
Furber: *ARM SoC Architecture* [Fur00]

- Aufteilen der Adresse in zwei Hälften
- $\overline{ras}$ „row address strobe“ wählt „Wordline“
  $\overline{cas}$ „column address strobe“ –"– „Bitline“
- je ein $2^{(n/2)}$-bit Decoder/Mux statt ein 2^n-bit Decoder

SRAM: statisches RAM

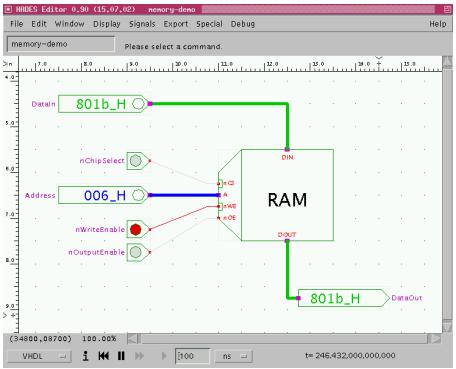
- Inhalt bleibt gespeichert solange Betriebsspannung anliegt
- *sechs-Transistor* Zelle zur Speicherung
  - weniger Platzverbrauch als Latches/Flipflops
  - kompakte Realisierung in CMOS-Technologie (s.u.)
  - zwei rückgekoppelte Inverter zur Speicherung
  - zwei n-Kanal Transistoren zur Anbindung an die Bitlines
- schneller Zugriff: Einsatz für Caches
- deutlich höherer Platzbedarf als DRAMs

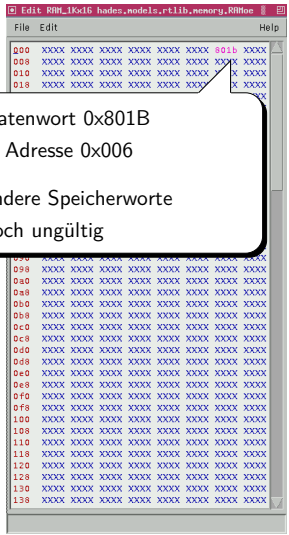
SRAM: Sechs-Transistor Speicherstelle („6T“)



[Hen] Hades Demo: `05-switched/40-cmos/sramcell`

SRAM: Hades Demo





Datenwort 0x801B
in Adresse 0x006

andere Speicherworte
noch ungültig

- nur aktiv wenn $nCS = 0$ (*chip select*)
- Schreiben wenn $nWE = 0$ (*write enable*)
- Ausgabe wenn $nOE = 0$ (*output enable*)

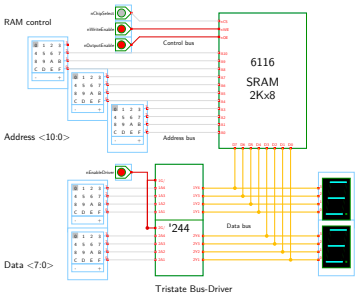
[Hen] Hades Demo: `50-rtlib/40-memory/ram`

SRAM: Beispiel IC 6116

- integrierte Schaltung, 16 Ki bit Kapazität
- Organisation als 2 Ki Worte mit je 8-bit

- 11 Adresseingänge (A10 . . . A0)
- 8 Anschlüsse für Tristate Daten-Eingang/-Ausgang
- 3 Steuersignale
  - $\overline{CS}$ chip-select: Speicher nur aktiv wenn $\overline{CS} = 0$
  - $\overline{WE}$ write-enable: Daten an gewählte Adresse schreiben
  - $\overline{OE}$ output-enable: Inhalt des Speichers ausgeben

- Hades-Demo zum Ausprobieren [Hen]
  - Hades Demo: `40-memories/40-ram/demo-6116`
  - Hades Demo: `40-memories/40-ram/two-6116`



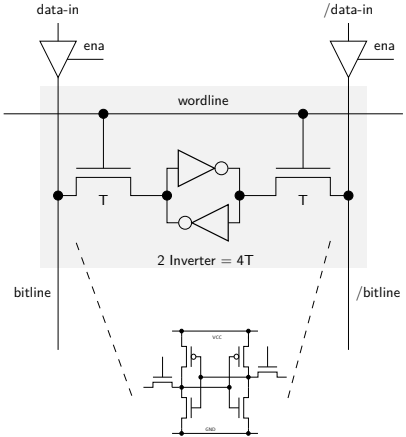
DRAM: dynamisches RAM

- Information wird in winzigen Kondensatoren gespeichert
- pro Bit je ein Transistor und Kondensator

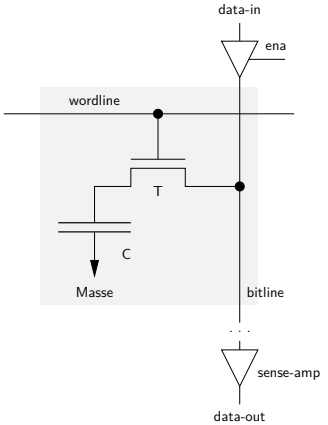
- jeder Lesezugriff entlädt den Kondensator
- *Leseverstärker* zur Messung der Spannung auf der Bitline
  Schwellwertvergleich zur Entscheidung logisch 0/1

– Information muss anschließend neu geschrieben werden
– auch ohne Lese- oder Schreibzugriff ist regelmäßiger *Refresh* notwendig,
  wegen Selbstentladung (Millisekunden)
– $10\times$ langsamer als SRAM
\+ DRAM für hohe Kapazität optimiert, minimaler Platzbedarf

SRAM vs. DRAM



- 6 Transistoren/bit
- statisch (kein refresh)
- schnell



- 1 Transistor/bit
- $C = 5\,\mathrm{fF} \approx 47\,000$ Elektronen
- langsam (sense-amp)

DRAM: Stacked- und Trench-Zelle

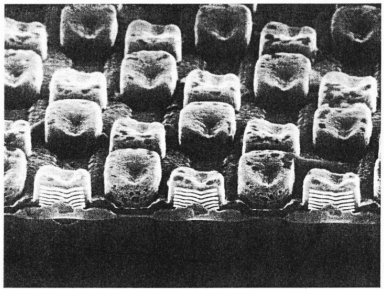
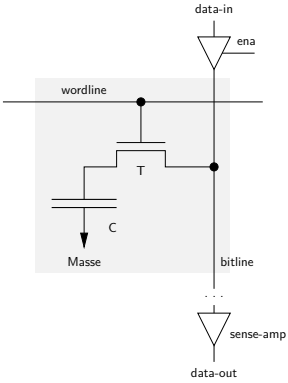


Abb. 7: Prototyp von Speicherzellen (Stapelkondensatoren) für zukünftige Speicherchips wie den Ein-Gigabit-Chip. Da für DRAM-Chips eine minimale Speicherkapazität von 25 fF notwendig ist, bringt es erhebliche Platzvorteile, die Kondensatorelemente vertikal übereinander zu stapeln. Die Dicke der Schichten beträgt etwa 50 nm. (Foto: Siemens)

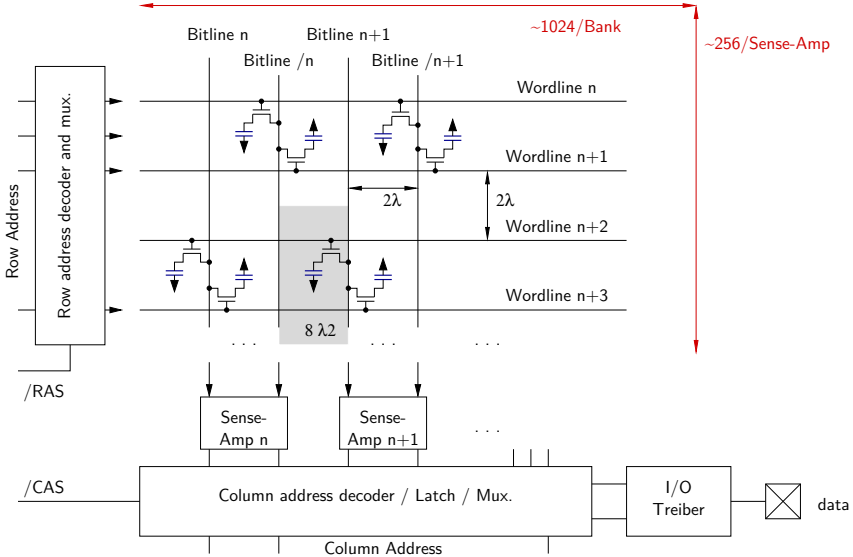
Siemens 1 Gbit DRAM



IBM CMOS-6X embedded DRAM

- zwei Bauformen: „stacked“ und „trench“
- Kondensatoren
  - möglichst kleine Fläche
  - Kapazität gerade ausreichend

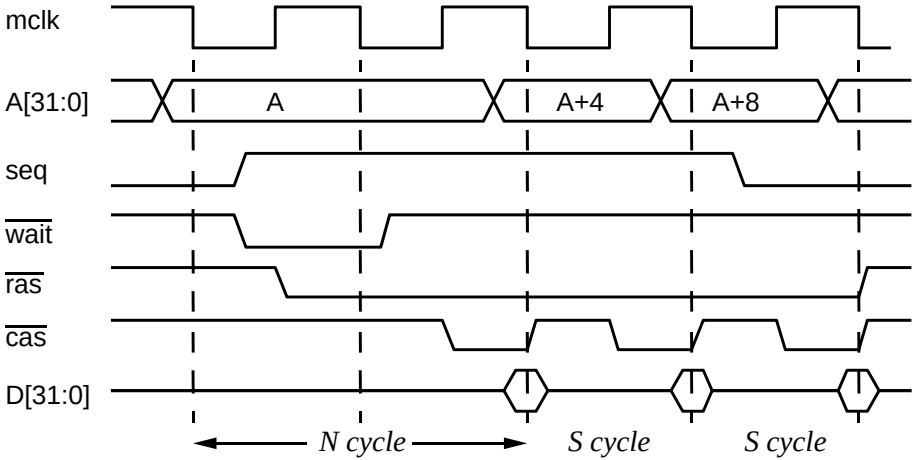
DRAM: Layout


~1024/Bank
~256/Sense-Amp
Bitline n
Bitline n+1
Bitline /n
Bitline /n+1
Wordline n
Wordline n+1
Wordline n+2
Wordline n+3
2λ
2λ
8 λ2
Row Address
Row address decoder and mux.
/RAS
Sense-
Amp n
Sense-
Amp n+1
/CAS
Column address decoder / Latch / Mux.
I/O
Treiber
data
Column Address

DRAM: Varianten

- veraltete Varianten
  - FPM: *fast-page mode*
  - EDO: *extended data-out*
  - ...
- heute gebräuchlich
  - SDRAM: Ansteuerung synchron zu Taktsignal
  - DDR-SDRAM: *double-data rate* Ansteuerung wie SDRAM
    Daten werden mit steigender und fallender Taktflanke übertragen
  - DDR2 ... DDR5: Varianten mit höherer Taktrate ... DDR6 ×2
    aktuelle Übertragungsraten bis 70,4 GByte/sec pro Speicherkanal
  - GDDR3 ... GDDR6X (*Graphics DDR*) ... GDDR7×1,5
    derzeit bis 168 GByte/sec
  - HBM ... HBM3E (*High Bandwidth Memory*)
    derzeit bis 1 254 GByte/sec

SDRAM: Lesezugriff auf sequenzielle Adressen



[Fur00]

Speicherorganisation

- Adressierung
- Wortbreite, Speicherkapazität
- „Big Endian“ / „Little Endian“
- „Alignment“
- „Memory-Map“
- Beispiel: PC mit Windows

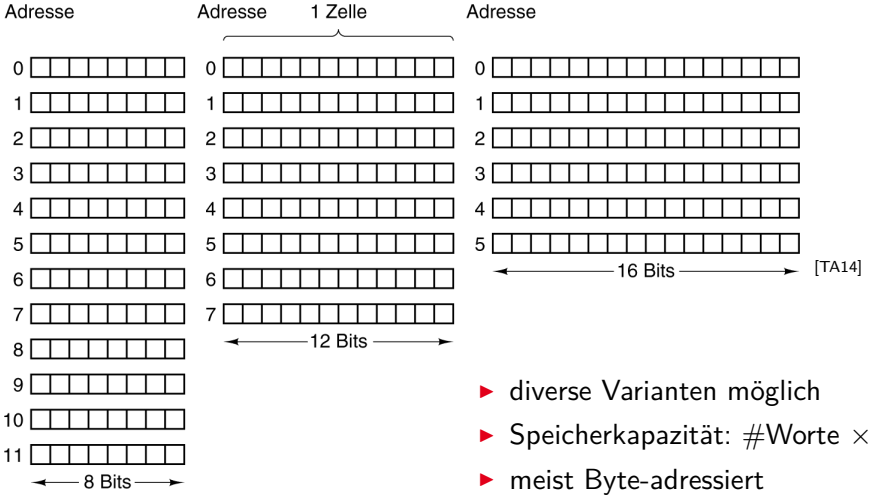
- spätere Themen
  - Cache-Organisation für schnelleren Zugriff
  - Virtueller Speicher für Multitasking
  - Synchronisation in Multiprozessorsystemen (z.B. MESI-Protokoll)

Aufbau und Adressierung des Speichers

- Abspeichern von Zahlen, Zeichen, Strings?
  - kleinster Datentyp üblicherweise ein Byte (8-bit)
  - andere Daten als Vielfache: 16-bit, 32-bit, 64-bit ...
- Organisation und Adressierung des Speichers?
  - Adressen typisch in Bytes angegeben
  - erlaubt Adressierung einzelner ASCII-Zeichen usw.
- aber Maschine/Prozessor arbeitet wortweise
- Speicher daher ebenfalls wortweise aufgebaut
- typischerweise 32-bit oder 64-bit

Hauptspeicherorganisation

3 Organisationsformen eines 96-bit Speichers: 12×8, 8×12, 6×16 Bits



[TA14]

- diverse Varianten möglich
- Speicherkapazität: #Worte $\times$ #Bits/Wort
- meist Byte-adressiert

Wortbreite

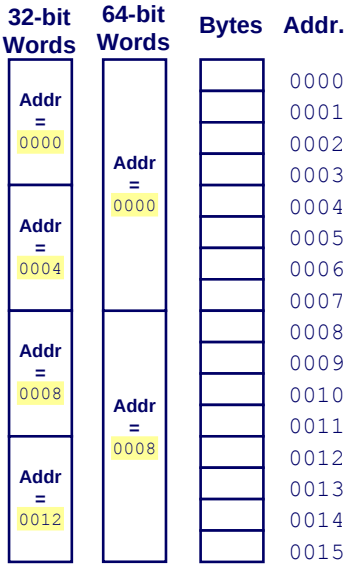
- Speicherwortbreiten historisch wichtiger Computer

Computer	Bits/Speicherzelle
Burroughs B1700	1
IBM PC	8
DEC PDP-8	12
IBM 1130	16
DEC PDP-15	18
XDS 940	24
Electrologica X8	27
XDS Sigma 9	32
Honeywell 6180	36
CDC 3600	48
CDC Cyber	60

- heute dominieren 8/16/32/64-bit Systeme
- erlaubt 8-bit ASCII, 16-bit Unicode, 32-/64-bit Floating-Point
- Beispiel x86: „byte“, „word“, „double word“, „quad word“

Wort-basierte Organisation des Speichers

- Speicher Wort-orientiert
- Adressierung Byte-orientiert
  - die Adresse des ersten Bytes im Wort
  - Adressen aufeinanderfolgender Worte unterscheiden sich um 4 (32-bit Wort) oder 8 (64-bit)
  - Adressen normalerweise Vielfache der Wortlänge
  - verschobene Adressen „in der Mitte“ eines Worts oft unzulässig



[BO15]

Datentypen auf Maschinenebene

- gängige Prozessoren unterstützen mehrere Datentypen
- entsprechend der elementaren Datentypen in C, Java ...
- `void*` ist ein **Pointer** (Referenz, Speicheradresse)
- Beispiel für die Anzahl der Bytes:

C Datentyp	DEC Alpha	typ. 32-bit	Intel IA-32 (x86)
int	4	4	4
long int	8	4	4
char	1	1	1
short	2	2	2
float	4	4	4
double	8	8	8
long double	8	8	10/12
void *	8	4	4

Datentypen auf Maschinenebene (cont.)

Abhängigkeiten (!)

- Prozessor
- Betriebssystem
- Compiler

segment word size	16 bit			32 bit						64 bit			
compiler	Microsoft	Borland	Watcom	Microsoft	Intel Windows	Borland	Watcom	Gnu, Clang	Intel Linux	Microsoft	Intel Windows	Gnu, Clang	Intel Linux
bool	2	1	1	1	1	1	1	1	1	1	1	1	1
char	1	1	1	1	1	1	1	1	1	1	1	1	1
wchar_t		2		2	2	2	2	2	2	2	2	4	4
short int	2	2	2	2	2	2	2	2	2	2	2	2	2
int	2	2	2	4	4	4	4	4	4	4	4	4	4
long int	4	4	4	4	4	4	4	4	4	4	4	8	8
int64_t				8	8			8	8	8	8	8	8
enum (typical)	2	2	1	4	4	4	4	4	4	4	4	4	4
float	4	4	4	4	4	4	4	4	4	4	4	4	4
double	8	8	8	8	8	8	8	8	8	8	8	8	8
long double	10	10	8	8	16	10	8	12	12	8	16	16	16
__m64				8	8			8	8		8	8	8
__m128				16	16			16	16	16	16	16	16
__m256				32	32			32	32	32	32	32	32
__m512				64	64			64	64	64	64	64	64
pointer	2	2	2	4	4	4	4	4	4	8	8	8	8
far pointer	4	4	4										
function pointer	2	2	2	4	4	4	4	4	4	8	8	8	8
data member pointer (min)	2	4	6	4	4	8	4	4	4	4	4	8	8
data member pointer (max)		4	6	12	12	8	12	4	4	12	12	8	8
member function pointer (min)	2	12	6	4	4	12	4	8	8	8	8	16	16
member function pointer (max)		12	6	16	16	12	16	8	8	24	24	16	16

`www.agner.org/optimize/calling_conventions.pdf`

Byte-Order

- *Wie sollen die Bytes innerhalb eines Wortes angeordnet werden?*
- Speicher wort-basiert $\Leftrightarrow$ Adressierung byte-basiert

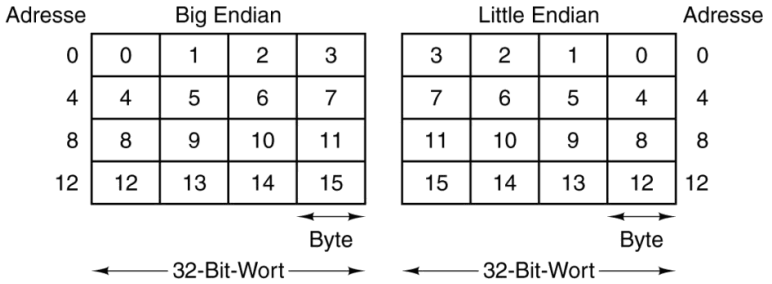
Zwei Möglichkeiten / Konventionen:

- **Big Endian**: Sun, Mac usw.
  das MSB (*most significant byte*) hat die kleinste Adresse
  das LSB (*least significant byte*) hat die höchste –"–

- **Little Endian**: Alpha, x86
  das MSB hat die höchste, das LSB die kleinste Adresse

satirische Referenz auf Gulliver's Reisen (Jonathan Swift)

Big- vs. Little Endian



[TA14]

- Anordnung einzelner Bytes in einem Wort (hier 32 bit)
  - Big Endian ($n \ldots n+3$): MSB $\ldots$ LSB „String“-Reihenfolge
  - Little Endian ($n \ldots n+3$): LSB $\ldots$ MSB „Zahlen“-Reihenfolge
- beide Varianten haben Vor- und Nachteile
- ggf. Umrechnung zwischen beiden Systemen notwendig

Byte-Order: Beispiel

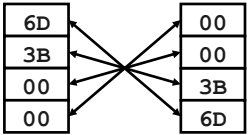

```
     int A =  15213;
     int B = -15213;
long int C =  15213;
```

Dezimal: 15213

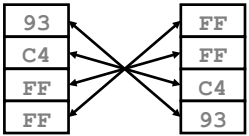
Binär: 0011 1011 0110 1101

Hex: 3 B 6 D

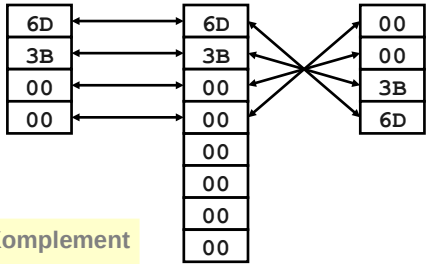
Linux/Alpha A **Sun A**



Linux/Alpha B **Sun B**



Linux C **Alpha C** **Sun C**



2-Komplement

Big Endian

Little Endian

[BO15]

Byte-Order: Beispiel Datenstruktur


```
/* JimSmith.c - example record for byte-order demo */

typedef struct employee {
  int      age;
  int   salary;
  char    name[12];
} employee_t;

static employee_t jimmy = {
  23,              // 0x0017
  50000,           // 0xc350
  "Jim Smith",     // J=0x4a i=0x69 usw.
};
```

Byte-Order: Beispiel x86 und SPARC


```
tams12> objdump -s JimSmith.x86.o
JimSmith.x86.o:     file format elf32-i386

Contents of section .data:
 0000 17000000 50c30000 4a696d20 536d6974  ....P...Jim Smit
 0010 68000000                             h...

tams12> objdump -s JimSmith.sparc.o
JimSmith.sparc.o:     file format elf32-sparc

Contents of section .data:
 0000 00000017 0000c350 4a696d20 536d6974  .......PJim Smit
 0010 68000000                             h...
```

Netzwerk Byte-Order

- Byte-Order muss bei Datenübertragung zwischen Rechnern berücksichtigt und eingehalten werden

- Internet-Protokoll (IP) nutzt ein Big Endian Format

$\Rightarrow$ auf x86-Rechnern (Little Endian) müssen alle ausgehenden und ankommenden Datenpakete umgewandelt werden

- zugehörige Hilfsfunktionen / Makros in `netinet/in.h`
  - inaktiv auf Big Endian, **byte-swapping** auf Little Endian
  - `ntohl(x)`: network-to-host-long
  - `htons(x)`: host-to-network-short
  - ...

Beispiel: Byte-Swapping *network to/from host*

Linux: /usr/include/bits/byteswap.h (distributionsabhängig)

```
...
/* Swap bytes in 32 bit value.  */
#define __bswap_32(x) \
  ((((x) & 0xff000000) >> 24) | (((x) & 0x00ff0000) >>  8) |\
   (((x) & 0x0000ff00) <<  8) | (((x) & 0x000000ff) << 24))
...
```

Linux: /usr/include/netinet/in.h

```
...
#  if __BYTE_ORDER == __LITTLE_ENDIAN
#   define ntohl(x) __bswap_32 (x)
#   define ntohs(x) __bswap_16 (x)
#   define htonl(x) __bswap_32 (x)
#   define htons(x) __bswap_16 (x)
#  endif
...
```

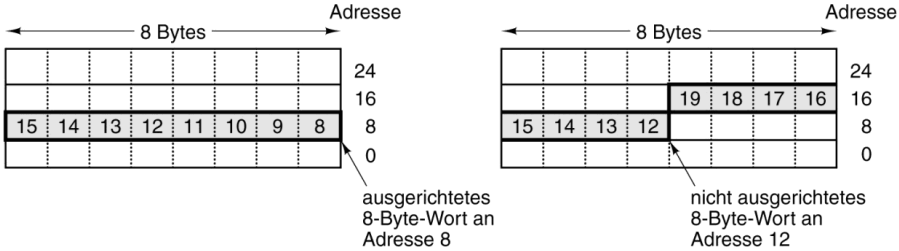
Programm zum Erkennen der Byte-Order

- Programm gibt Daten byteweise aus
- C-spezifische Typ- (Pointer-) Konvertierung
- Details: Bryant, O'Hallaron [BO15], 2.1.4 (Abb. 2.3, 2.4)

```
void show_bytes( byte_pointer start, int len ) {
  int i;
  for( i=0; i < len; i++ ) {
    printf( " %.2x", start[i] );
  }
  printf( "\n" );
}

void show_double( double x ) {
  show_bytes( (byte_pointer) &x, sizeof( double ));
}
  ...
```

„Misaligned“ Zugriff



[TA14]

- Speicher wird (meistens) Byte-weise adressiert, aber Zugriffe lesen/schreiben jeweils ein ganzes Speicherwort
- Beispiel: 8-Byte-Wort in Little Endian Speicher
  - *„aligned“* bezüglich Speicherwort
  - *„non aligned“* an Byte-Adresse 12

⇒ was passiert bei „krummen“ (*misaligned*) Adressen?

- automatische Umsetzung auf mehrere Zugriffe (x86)
- Programmabbruch (SPARC)

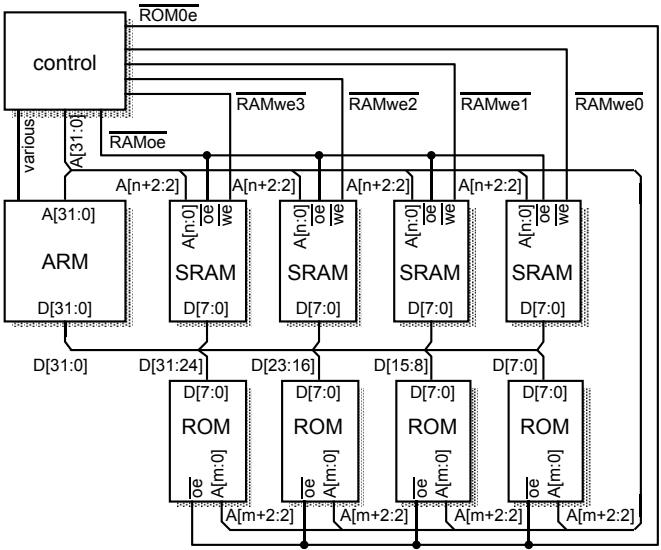
Memory Map

- CPU kann im Prinzip alle möglichen Adressen ansprechen
- in der Regel: kein voll ausgebauter Speicher
  32 bit Adresse entsprechen 4 GiB Hauptspeicher, 64 bit ...

⇒ „Memory Map“

- Adressdecoder als Hardwareeinheit
  - Aufteilung in *read-write*- und *read-only*-Bereiche
  - ROM zum Booten notwendig
  - Read-only in *eingebetteten Systemen*: Firmware, OS, Programme
  - zusätzliche Speicherbereiche für „memory mapped“ I/O
- Adressabbildung in Betriebssystemen (Windows, Linux etc.)
  - Zuordnung von Adressen zu „realem“ Speicher
  - alle Hardwarekomponenten (+ Erweiterungskarten)
    Ein-/Ausgabekanäle
    Interrupts
  - Verwaltung über *Treiber*

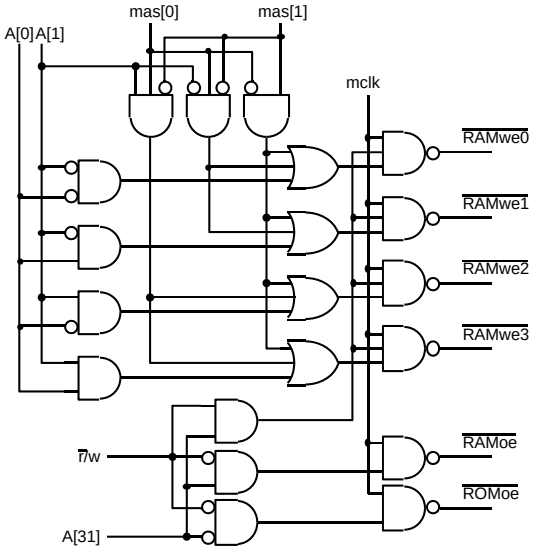
Adressabbildung Hardware: ARM



32-bit ARM Prozessor
4 × 8-bit SRAMs
4 × 8-bit ROMs

[Fur00]

Adressabbildung Hardware: ARM (cont.)



Adressdecoder Hardware

[Fur00]

Memory Map: typ. 16-bit System

- 16-bit erlaubt 64K Adressen: 0x0000 . . . 0xFFFF

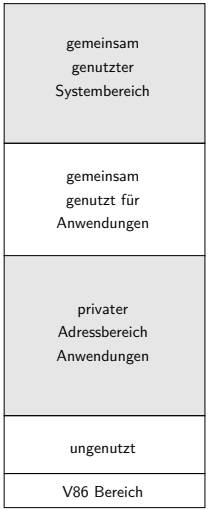
- ROM-Bereich für Boot / Betriebssystemkern
- RAM-Bereich für Betriebssystem (z.B. Interrupt-Tabelle)
- RAM-Bereich für Hauptspeicher (Programme und Daten)

- I/O-Bereiche für serielle / parallele Schnittstellen
- I/O-Bereiche für weitere Schnittstellen

Demo und Beispiel: im Praktikum RSB (64-042)

Adressabbildung in Betriebssystemen

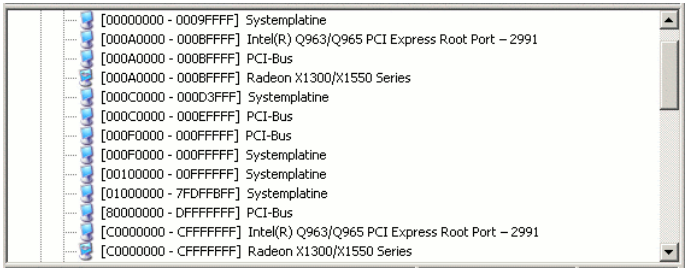
- Windows 95 (32-bit System: 4 GiB Adressraum)



- Kernel, Treiber usw. im oberen 1 GiB-Bereich
- Systembibliotheken (DLLs)
- 2 GiB für Anwendungen
- DOS-Bereich (real-mode): für Boot + Geräte die „unteren“ 20-bit

Adressabbildung in Betriebssystemen (cont.)

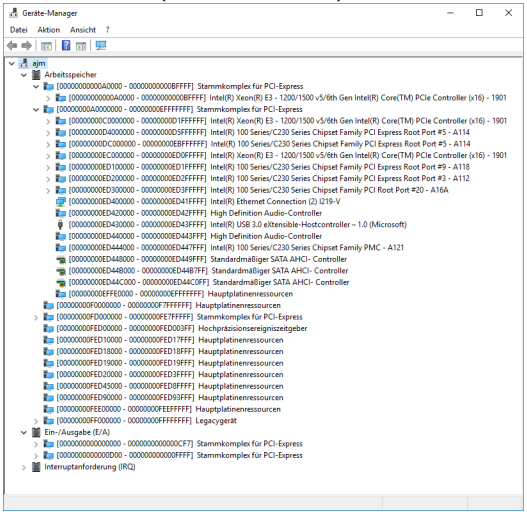
- Windows 95 (32-bit System)
  - 32-bit Adressen, 4 GiByte Adressraum
  - Aufteilung 2 GiB für Programme, obere 1+1 GiB für Windows
  - unabhängig vom physisch vorhandenen Speicher
  - Beispiel der Zuordnung, diverse Bereiche für I/O reserviert



  - x86 I/O-Adressraum gesamt nur 64 KiByte
  - je nach Zahl der I/O-Geräte evtl. fast voll ausgenutzt
  - Adressen vom BIOS zugeteilt

Adressabbildung in Betriebssystemen (cont.)

- Windows 10 (64-bit System)

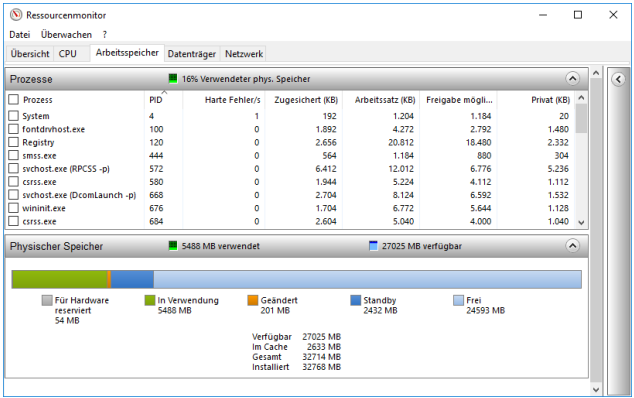


⇒ Adressbereiche

Adressabbildung in Betriebssystemen (cont.)

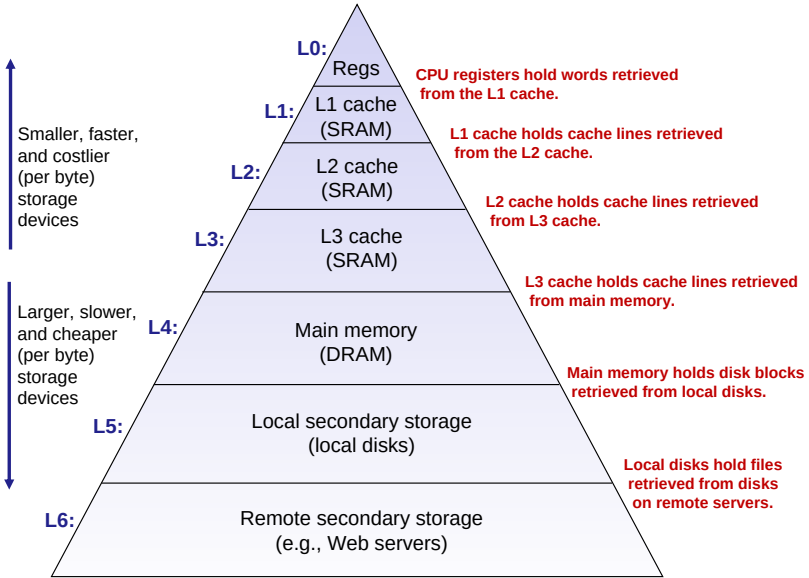
mehrstufige Abbildung:

1. alle Hardwarekomponenten, Ein-/Ausgabeeinheiten und Interrupts $\Rightarrow$ Adressbereiche
2. Adressbereiche $\Rightarrow$ physisch vorhandener Speicher



- Linux siehe: `free`, `vmstat`, `dmidecode`, `hwinfo`, `top`, `lshw`, ...

Demnächst: Speicherhierarchie



[BO15] später mehr . . .

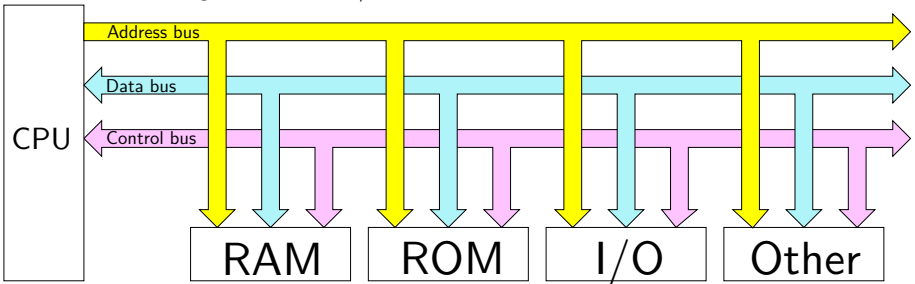
Speicher: die wesentliche Komponente eines Rechners

- Speicher ist nicht unbegrenzt
  - muss zugeteilt und verwaltet werden
  - viele Anwendungen werden vom Speicher dominiert
- besondere Sorgfalt beim Umgang mit Speicher
  - Fehler sind besonders gefährlich und schwer zu Debuggen
  - Auswirkungen sind sowohl zeitlich als auch räumlich entfernt
- Speicherleistung ist nicht konstant
  von Größe des Speichers, Datenstrukturen und -zugriffen ahhängig
- Wechselwirkungen: Speichersystem $\Leftrightarrow$ Programme
  - Cacheorganisation und Virtual-Memory beeinflussen Performanz/Programmleistung
  - Anpassung des Programms an das Speichersystem kann Laufzeitverhalten verbessern!

$\rightarrow$ siehe **??** *Rechnerarchitektur II – Speicherhierarchie*

Bussysteme

- **Bus:** elektrische (und logische) Verbindung
  - mehrere Geräte
  - mehrere Blöcke innerhalb einer Schaltung
- Bündel aus Daten- und Steuersignalen
  - Kernkomponenten (CPU, Speicher ...) miteinander verbinden
  - Verbindungen zu den Peripherie-Bausteinen
  - Verbindungen zu Systemmonitor-Komponenten
  - Verbindungen zwischen I/O-Controllern und -Geräten



Bussysteme (cont.)

- konzeptionell bidirektional
  - mehrere Quellen und mehrere Senken (lesende Zugriffe)
  - elektrische Realisierung: Tri-State-Treiber oder Open-Drain
- heute oft: Punkt-zu-Punkt Verbindungen
  - hohe Übertragungsraten technisch nicht anders möglich
  - logischer Bus (für Betriebssystem)

▸ Exkurs: Tristate

Unterscheidungskriterien

- Bus-Arbitrierung: wer darf, wann, wie lange senden?
  - Master-Slave
  - gleichberechtigte Knoten, Arbitrierungsprotokolle
- synchron: mit globalem Taktsignal vom „Master“-Knoten
  asynchron: Wechsel von Steuersignalen löst Ereignisse aus
- viele Typen standardisiert, mit sehr unterschiedlichen Anforderungen
  - High-Performance (= Datendurchsatz)

Bussysteme (cont.)

- einfaches Protokoll, billige Komponenten
- Multi-Master-Fähigkeit, zentrale oder dezentrale Arbitrierung
- Echtzeitfähigkeit, Daten-Streaming
- wenig Leitungen bis zu Zweidraht-Bussen: I^2C, SPI, System-Management-Bus ...
- lange Leitungen: EIA-485, RS-232, Ethernet ...
- Funkmedium: WLAN, Bluetooth ... (logische Verbindung)

Bus: Mikroprozessorsysteme

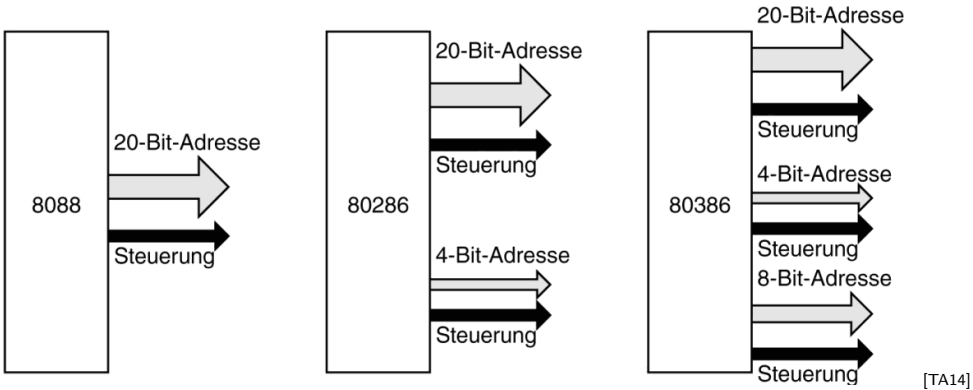
typisches n-bit Mikroprozessor-System:

- n Adress-Leitungen, also Adressraum 2^n Bytes — Adressbus
- n Daten-Leitungen — Datenbus

- Steuersignale — Control
  - clock: Taktsignal
  - read/write: Lese-/Schreibzugriff (aus Sicht des Prozessors)
  - wait: Wartezeit/-zyklen für langsame Geräte
  - ...

- um Leitungen zu sparen: teilweise gemeinsam genutzte Leitungen sowohl für Adressen als auch Daten $\Rightarrow$ zusätzliches Steuersignal zur Auswahl Adressen/Daten

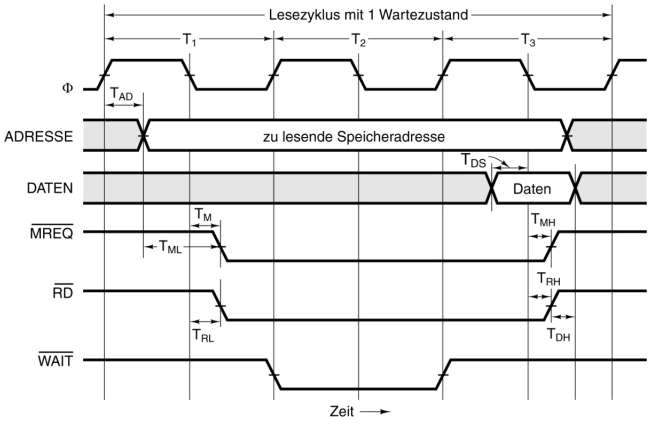
Adressbus: Evolution beim Intel x86



[TA14]

- 20-bit: 1 MiByte Adressraum
  24-bit: 16 MiByte
  32-bit: 4 GiByte
- alle Erweiterungen abwärtskompatibel
- 64-bit Architekturen: 48-, 56-, 64-bit Adressraum

Synchroner Bus: Lesezugriff mit Timing



[TA14]

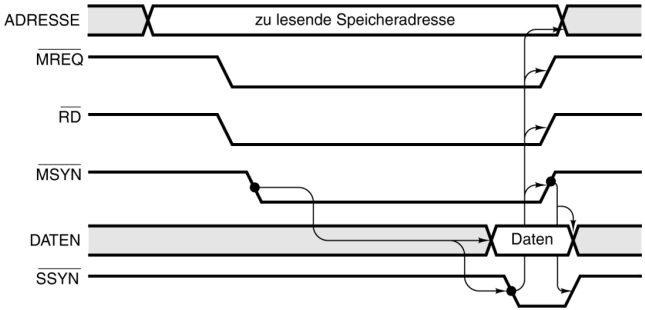
- alle Zeiten über Taktsignal Φ gesteuert
- $\overline{MREQ}$-Signal zur Auswahl Speicher oder I/O-Geräte
- $\overline{RD}$ signalisiert Lesezugriff
- Wartezyklen, solange der Speicher $\overline{WAIT}$ aktiviert

Synchroner Bus: Timing (cont.)

- typische Parameter

Symbol		[ns] Min	Max
T_{AD}	Adressausgabeverzögerung		4
T_{ML}	Adresse ist vor $\overline{MREQ}$ stabil	2	
T_M	$\overline{MREQ}$-Verzögerung nach fallender Flanke von Φ in T_1		3
T_{RL}	*RD*-Verzögerung nach fallender Flanke von Φ in T_1		3
T_{DS}	*Setup*-Zeit vor fallender Flanke von Φ	2	
T_{MH}	$\overline{MREQ}$-Verzögerung nach fallender Flanke von Φ in T_3		3
T_{RH}	$\overline{RD}$-Verzögerung nach fallender Flanke von Φ in T_3		3
T_{DH}	*Hold*-Zeit nach der Deaktivierung von $\overline{RD}$	0	

Asynchroner Bus: Lesezugriff



[TA14]

- Steuersignale $\overline{MSYN}$: Master fertig
  $\overline{SSYN}$: Slave fertig
- Handshake zwischen Sender und Empfänger (hier vier Phasen)
- flexibler für Geräte mit stark unterschiedlichen Zugriffszeiten

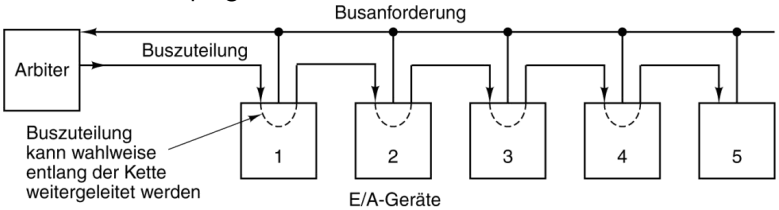
Bus Arbitrierung

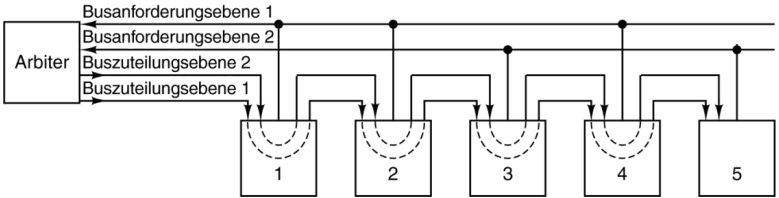
- mehrere Komponenten wollen Übertragung initiieren, aber nur ein Transfer zur Zeit
- der Bus Zugriff muss serialisiert werden

1. zentrale Arbitrierung
   - Arbiter gewährt Bus-Requests
   - Strategien
     - Prioritäten für verschiedene Geräte
     - „round-robin“ Verfahren
     - „Token“-basierte Verfahren
     - usw.
2. dezentrale Arbitrierung
   - protokollbasiert
   - Beispiel
     - Komponenten sehen ob Bus frei ist
     - wenn ja: Start der Übertragung; wenn nein: warten
     - zur Kollisionserkennung: Daten vom Bus lesen
     - bei Inkonsistenzen: Übertragung abbrechen und „später“ erneut versuchen

Bus Arbitrierung (cont.)

- I/O-Geräte oft höher priorisiert als die CPU
  - I/O-Zugriffe müssen schnell/sofort behandelt werden
  - Benutzerprogramm kann warten





[TA14]

Bus Bandbreite

- Menge an (Nutz-) Daten, die pro Zeiteinheit übertragen werden kann
- zusätzlicher Protokolloverhead ⇒ Brutto- / Netto-Datenrate

RS-232	50 bit/sec	...	460 Kbit/sec	Peripherie
I^2C	100 Kbit/sec (Std.)	...	3,4 Mbit/sec (High Speed)	
USB	1,5 Mbit/sec (1.x)	...	80 Gbit/sec (4.0v2)	
ISA	128 Mbit/sec	...		MB-Bus
PCI	1 Gbit/sec (2.0)	...	6,4 Gbit/sec (3.0)	
AGP	2,1 Gbit/sec (1x)	...	34,1 Gbit/sec (8x 64-bit)	
PCI Express	2,5 Gbit/sec (1.x)	...	1,9 Tbit/sec (7.0) ×16	
NVLink	640,0 Gbit/sec (1.0)	...	6,4 Tbit/sec (5.0)	GPUs
HyperTransport	25,6 Gbit/sec (1.0)	...	409,6 Gbit/sec (3.1)	CPUs
Infinity Fabric		...	4,1 Tbit/sec	

- `en.wikipedia.org/wiki/List_of_interface_bit_rates`

Beispiel: PCI-Bus

Peripheral Component Interconnect (Intel 1991)

- 33 MHz Takt — optional 66 MHz Takt
- 32-bit Bus-System — optional auch 64-bit
- gemeinsame Adress-/Datenleitungen
- Arbitrierung durch Bus-Master (die CPU)

- Abwärtskompatibilität
  - PCI-Bus als logische Verbindung (SW-Layer) zu Komponenten
  - technisch: PCIe

- Auto-Konfiguration
  - angeschlossene Geräte werden automatisch erkannt
  - eindeutige Hersteller- und Geräte-Nummern
  - Betriebssystem kann zugehörigen Treiber laden
  - automatische Zuweisung von Adressbereichen und IRQs

PCI-Bus: Peripheriegeräte

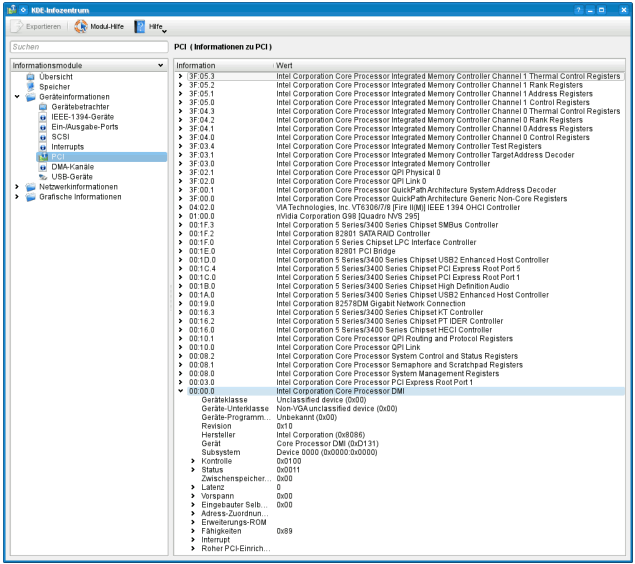

```
[maeder@tams165]~>lspci -v
00:00.0 Host bridge: Intel Corporation Sky Lake Host Bridge/DRAM Registers (rev 08)
        Subsystem: Dell Device 06dc
        Flags: bus master, fast devsel, latency 0
        Capabilities: <access denied>

00:02.0 VGA compatible controller: Intel Corporation Sky Lake Integrated Graphics (rev 07)
        Subsystem: Dell Device 06dc
        Flags: bus master, fast devsel, latency 0, IRQ 134
        Memory at e0000000 (64-bit, non-prefetchable) [size=16M]
        Memory at d0000000 (64-bit, prefetchable) [size=256M]
        I/O ports at f000 [size=64]
        Expansion ROM at <unassigned> [disabled]
        Capabilities: <access denied>
        Kernel driver in use: i915_bpo

00:04.0 Signal processing controller: Intel Corporation Device 1903 (rev 08)
        Subsystem: Dell Device 06dc
        Flags: bus master, fast devsel, latency 0, IRQ 16
        Memory at e1340000 (64-bit, non-prefetchable) [size=32K]
        Capabilities: <access denied>
        Kernel driver in use: proc_thermal

00:14.0 USB controller: Intel Corporation Device 9d2f (rev 21) (prog-if 30 [XHCI])
        Subsystem: Dell Device 06dc
        Flags: bus master, medium devsel, latency 0, IRQ 125
        Memory at e1330000 (64-bit, non-prefetchable) [size=64K]
        ...
```

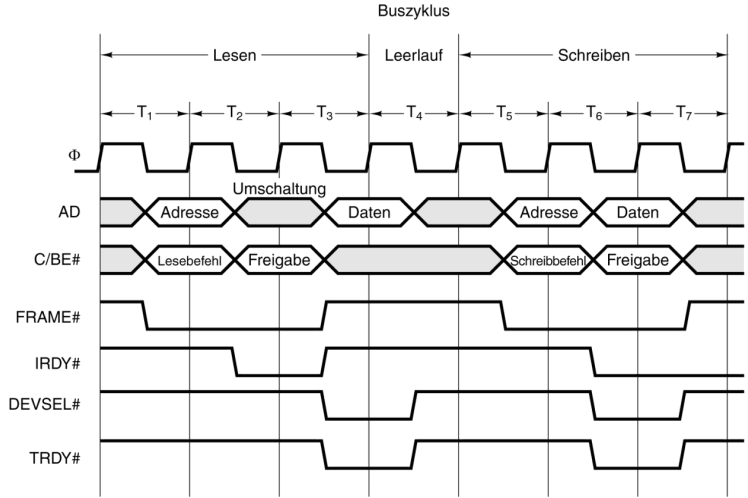
PCI-Bus: Peripheriegeräte (cont.)



PCI-Bus: Leitungen („mindestens“)

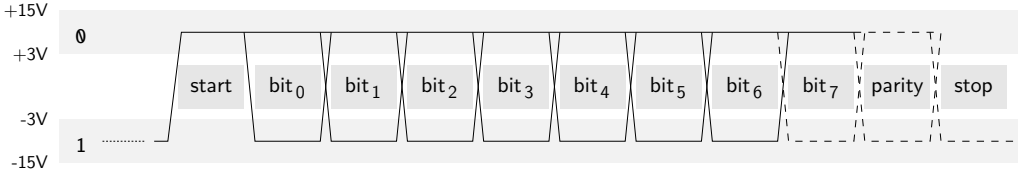
Signal	Leitungen	Master	Slave	Beschreibung
CLK	1			Takt (33 oder 66 MHz)
AD	32	×	×	Gemultiplexte Adress- und Datenleitungen
PAR	1	×		Adress- oder Datenparitätsbit
C/BE	4	×		Busbefehl/Bitmap für Byte Enable (zeigt gültige Datenbytes an)
FRAME#	1	×		Kennzeichnet, dass AD und C/BE aktiviert sind
IRDY#	1	×		Lesen: Master wird akzeptieren Schreiben: Daten liegen an
IDSEL	1	×		Wählt Konfigurationsraum statt Speicher
DEVSEL#	1		×	Slave hat seine Adresse decodiert und ist in Bereitschaft
TRDY#	1		×	Lesen: Daten liegen an Schreiben: Slave wird akzeptieren
STOP#	1		×	Slave möchte Transaktion sofort abbrechen
PERR#	1			Empfänger hat Datenparitätsfehler erkannt
SERR#	1			Adressparätätsfehler oder Systemfehler erkannt
REQ#	1			Bus-Arbitration: Anforderung des Busses
GNT#	1			–"– Zuteilung des Busses
RST#	1			Setzt das System und alle Geräte zurück

PCI-Bus: Transaktionen



[TA14]

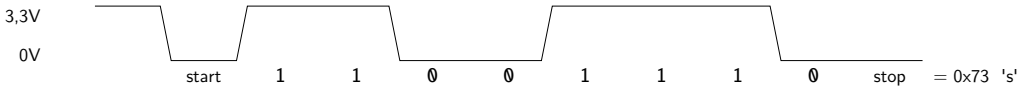
RS-232: Serielle Schnittstelle



- einfache serielle Datenübertragung
- asynchron (kein Taktsignal) Quelle beginnt Übertragung
- zu vereinbaren: Baudrate 110 . . . 19 200, 38 400, 115 200 bits/sec
  # Datenbits 5, 6, 7, 8
  # Stopbits 1, 2
  Parität none, odd, even
- min. 3 Leitungen: GND, TX, RX (Masse, Transmit, Receive)
  oft weitere Leitungen für erweitertes Handshake
- ursprünglich für Fernschreiber ($\pm$ 12V)

RS-232: Serielle Schnittstelle (cont.)

- später Anschluß von Terminals an Rechner, Datenübertragung (Modems)
  PC: Ein- und Ausgabe
- unterschiedliche Logikpegel (TTL, CMOS)
- Beispiel: 8/N/1 (8-bit, keine Parity, 1-Stopbit)

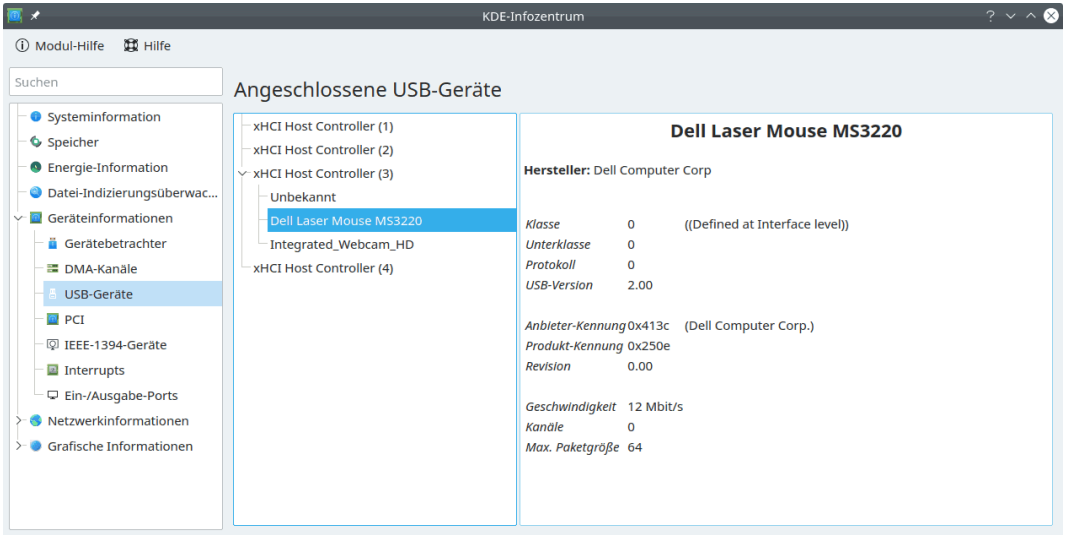


USB: Universal Serial Bus

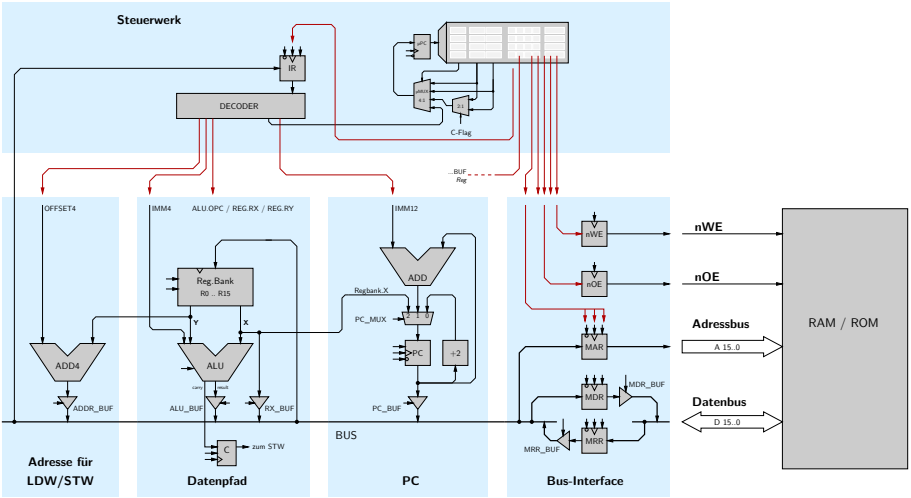
- 1996: serielle Datenübertragung von/zu Peripheriegeräten (12 Mb/s)
- differentielle Signale (störunanfällig), inzwischen 80 Gb/s
- Hot-plug fähig, Baumstruktur mit Hubs
- verschiedene Geräteklassen
  - Ein-/Ausgabegeräte: Tastatur, Maus, Drucker, Scanner, Audio
  - Massenspeicher: Festplatten, DVD etc.
  - Netzwerk
  - Monitor (DisplayPort), Grafikkarten
  - Dongles
- Energieversorgung: 0,5 W ... 240 W

```
maeder@tams180:~$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 002: ID 413c:250e Dell Computer Corp. Dell Laser Mouse MS3220
Bus 003 Device 003: ID 0c45:6a14 Microdia Integrated_Webcam_HD
Bus 003 Device 004: ID 8087:0033 Intel Corp.
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
```

USB: Universal Serial Bus (cont.)



Praktikum RSB: D·CORE



Ablaufsteuerung mit Mikroprogramm

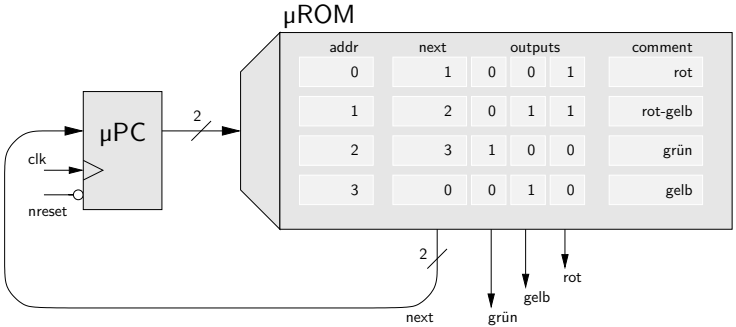
- Alternative zu direkt entworfenen Schaltwerken (δ-/λ-Schaltnetze aus Gattern)
- „Umprogrammierung“ durch Austasch des Mikroprogramms
  - Fehlerbehebung möglich

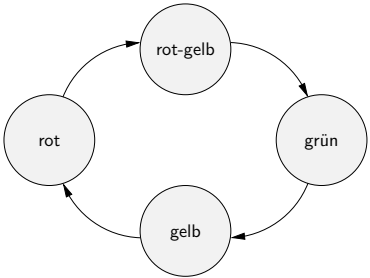
  $\Rightarrow$ Firmware für Prozessoren

- *Mikroprogrammzähler* μPC: Register für aktuellen Zustand
- μPC adressiert den Mikroprogrammspeicher μROM
- μROM konzeptionell in mehrere Felder eingeteilt
  - die verschiedenen Steuerleitungen
  - ein oder mehrere Felder für Folgezustand
  - ggf. zusätzliche Logik und Multiplexer zur Auswahl unter mehreren Folgezuständen
  - ggf. Verschachtelung und Aufruf von Unterprogrammen: „nanoProgramm“

- siehe „Praktikum Rechnerstrukturen und Betriebssysteme“

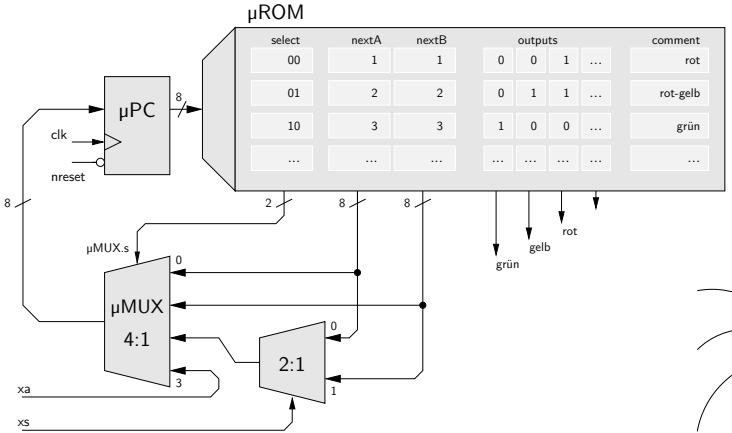
Mikroprogramm: Beispiel Ampel

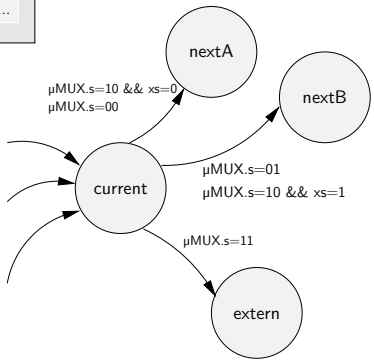




- μPC adressiert das μROM
- *next*-Ausgang liefert Folgezustand
- andere Ausgänge steuern die Schaltung
  = die Lampen der Ampel

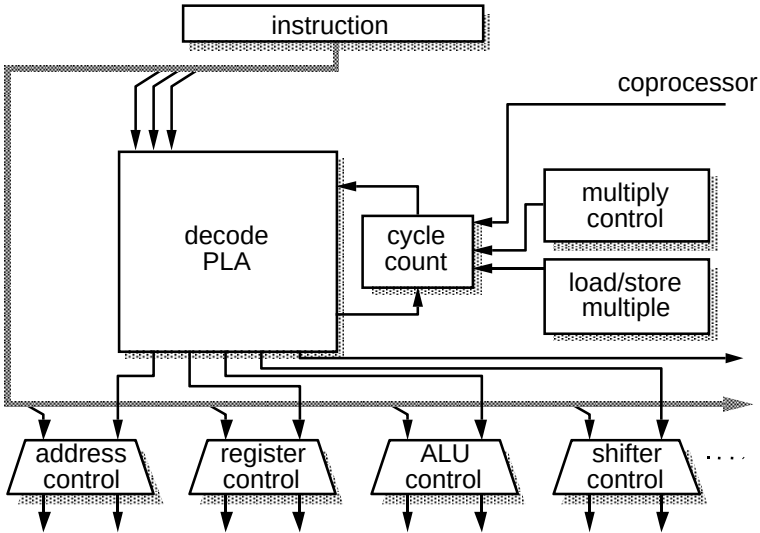
Mikroprogramm: Beispiel zur Auswahl des Folgezustands





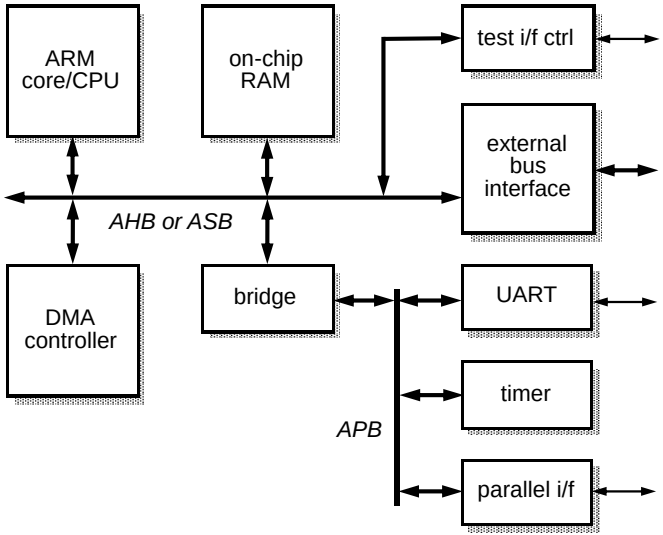
- Multiplexer erlaubt Auswahl des μPC Werts
- $nextA$, $nextB$ aus dem μROM, externer xa Wert
- xs Eingang für bedingte Sprünge

Mikroprogramm: Befehlsdecoder des ARM 7 Prozessors



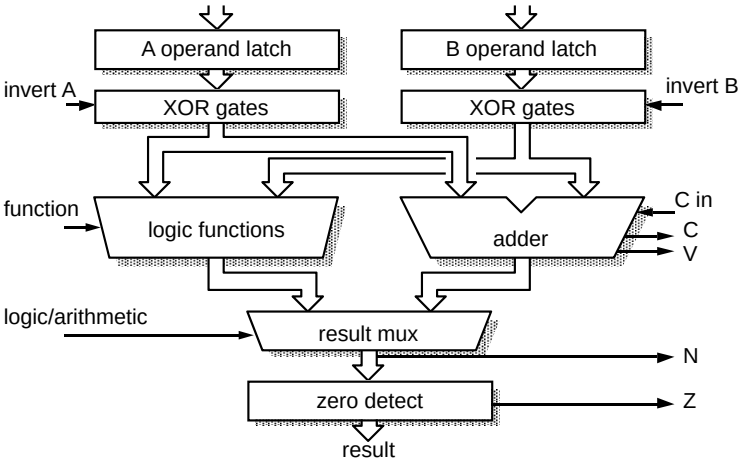
[Fur00]

typisches ARM SoC System



S. Furber: *ARM System-on-Chip Architecture* [Fur00]

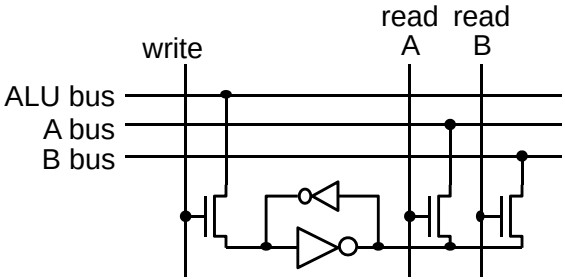
RT-Ebene: ALU des ARM 6 Prozessors



[Fur00]

- Register für die Operanden A und B
- Addierer und separater Block für logische Operationen

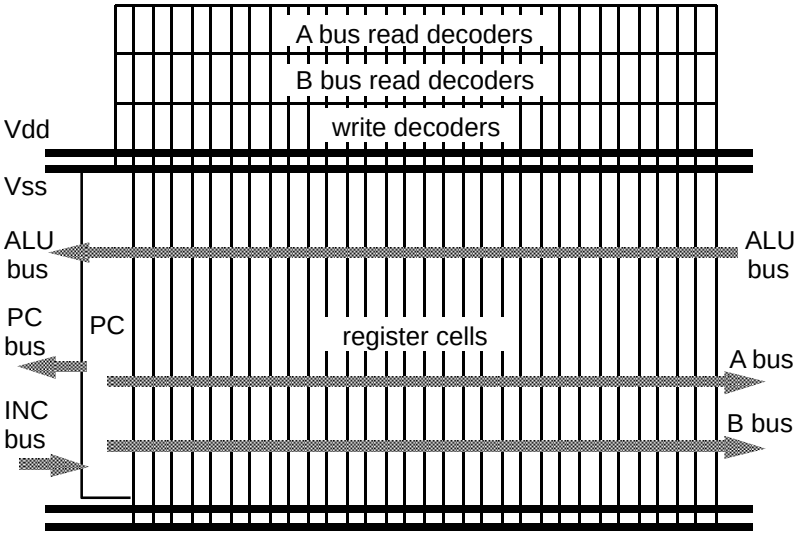
Multi-Port-Registerbank: Zelle



[Fur00]

- Prinzip wie 6T-SRAM: rückgekoppelte Inverter
- mehrere (hier zwei) parallele Lese-Ports
- mehrere Schreib-Ports möglich, aber kompliziert

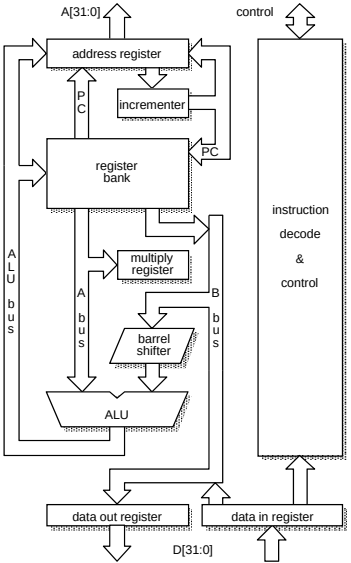
Multi-Port Registerbank: Floorplan/Chiplayout



[Fur00]

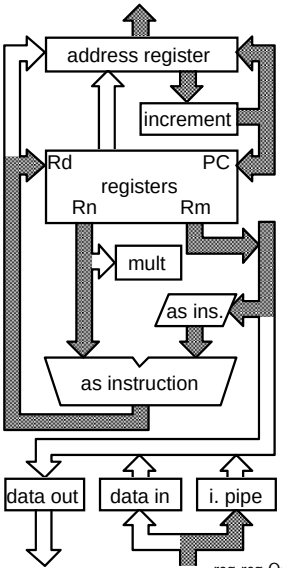
Kompletter Prozessor: ARM 3

- Registerbank (inkl. Program Counter)
- Inkrementer
- Adress-Register
- ALU, Multiplizierer, Shifter
- Speicherinterface (Data-In / -Out)
- Steuerwerk
- bis ARM 7: 3-stufige Pipeline
  fetch, *decode*, *execute*

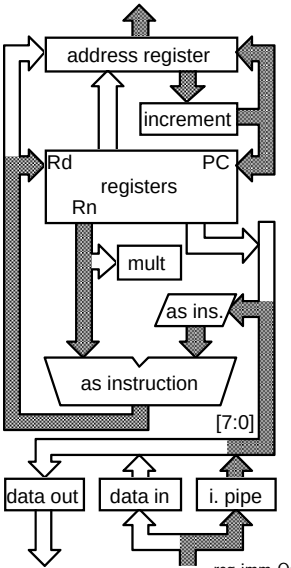


[Fur00]

ARM Datentransfer: Register-Operationen



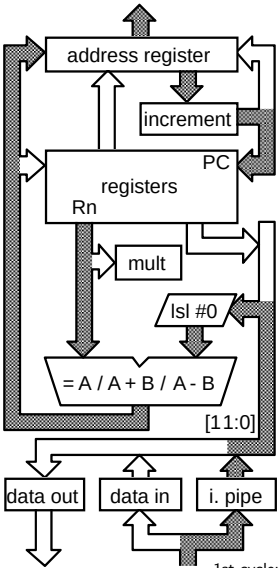
reg-reg Operation



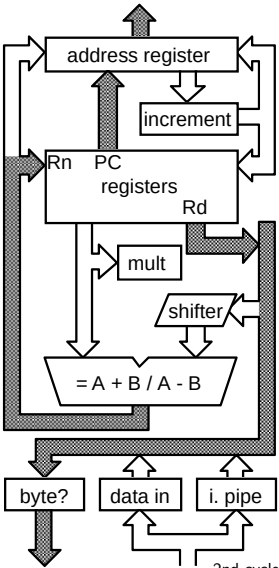
reg-imm Operation

[Fur00]

ARM Datentransfer: Store-Befehl

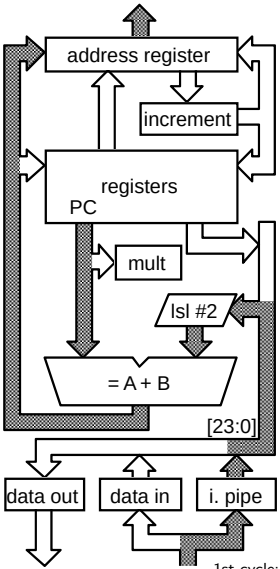


1st cycle: compute address

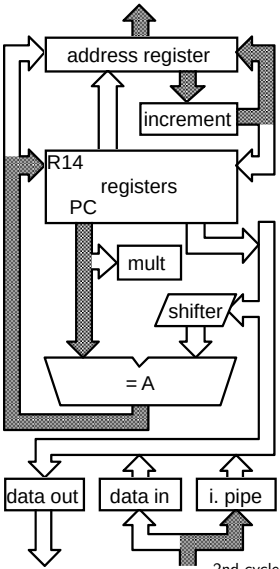


2nd cycle: store & auto-index [Fur00]

ARM Datentransfer: Funktionsaufruf/Sprungbefehl

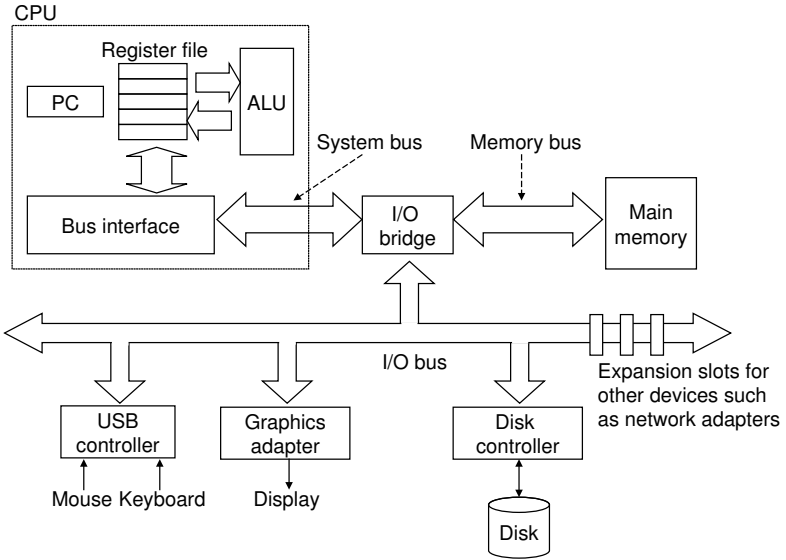


1st cycle: compute branch target



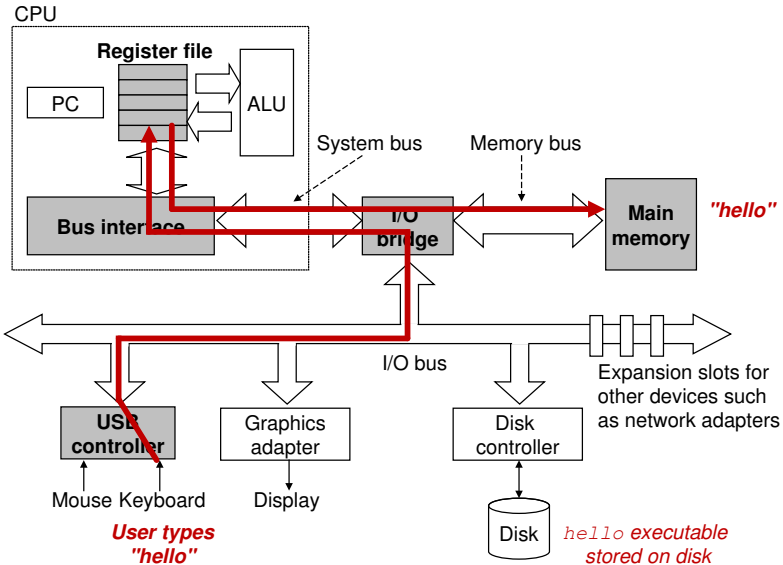
2nd cycle: save return address [Fur00]

Hardwareorganisation eines typischen PC



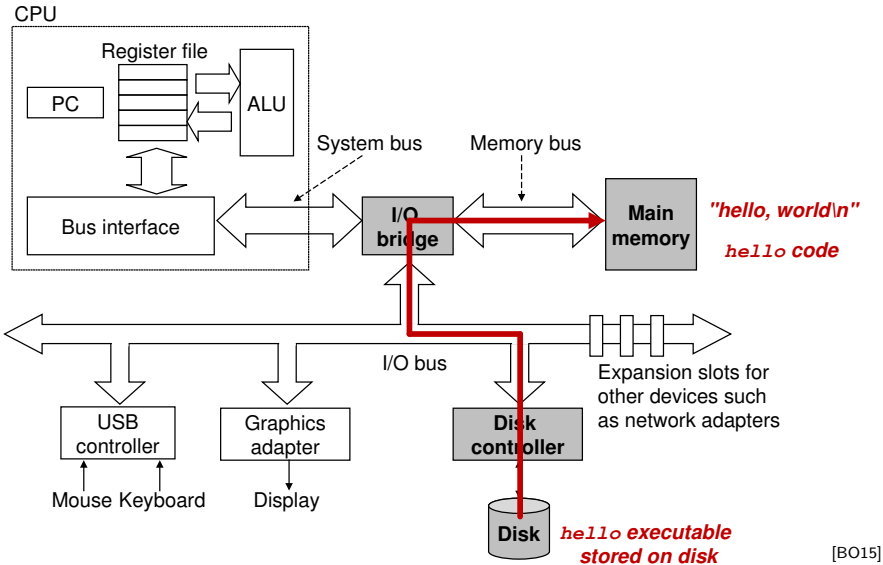
[BO15]

Programmausführung: 1. Benutzereingabe



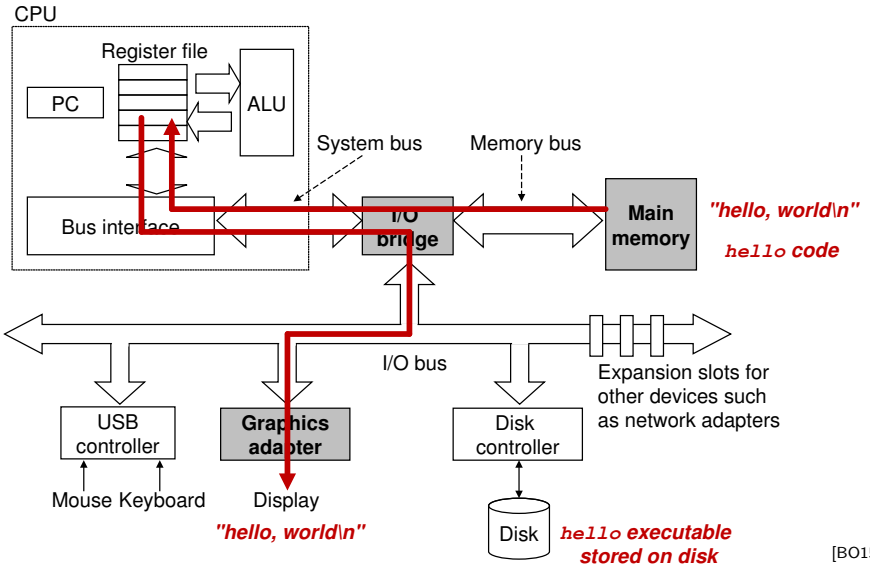
[BO15]

Programmausführung: 2. Programm laden



[BO15]

Programmausführung: 3. Programmlauf



[BO15]

Literatur

[BO15] Randal E. Bryant and David R. O'Hallaron.
Computer systems – A programmers perspective.
Pearson Education Limited, Harlow, third, global ed. edition, 2015.

[Fur00] Steve Furber.
ARM System-on-Chip Architecture.
Pearson Education Limited, Harlow, second edition, 2000.

[GK83] Daniel D. Gajski and Robert H. Kuhn.
Guest editors' introduction: New vlsi tools.
IEEE Computer, 16(12):11–14, December 1983.

[Hen] Norman Hendrich.
Hades — hamburg design system.
Lehrmaterial, Universität Hamburg, Fachbereich Informatik, AB TAMS.

Literatur (cont.)

[Mä11] Andreas Mäder.
Vorlesung: Rechnerarchitektur und mikrosystemtechnik.
Vorlesungsfolien, Universität Hamburg, Fachbereich Informatik, AB TAMS, 2011.

[PH20] David A. Patterson and John L. Hennessy.
Computer Organization and Design – The Hardware Software Interface – RISC-V Edition.
Morgan Kaufmann Publishers Inc., San Mateo, CA, second edition, 2020.

[PH22] David A. Patterson and John L. Hennessy.
Rechnerorganisation und Rechnerentwurf – Die Hardware/Software-Schnittstelle – MIPS Edition.
De Gruyter Oldenbourg, Berlin, sixth edition, 2022.

Literatur (cont.)

[TA14] Andrew S. Tanenbaum and Todd Austin.
Rechnerarchitektur – Von der digitalen Logik zum Parallelrechner.
Pearson Deutschland GmbH, Hallbergmoos, sixth edition, 2014.