Computersysteme 1

1. Hardware

BIOS = Basic Input Output System: Hardwarecheck, Simple Displayansteuerung
ALU = Arithmetic Logical Unit CPU = Central Processing Unit GPU = Graphical Processing Unit RAM = Random Access Memory GPR = General Purpose Register SPR = Special Purpose Register

1.1. Grafikkarte

Bestnandteile: Speicher, GPU, BIOS, Shnittstellen
VGA: R, G, B, Hsync, Vsync
Antialiasing: Berechnung der prozentualen Farbintensitäten bei Farbüberdeckung.
Front- und Backbuffer für Flickerfreie Darstellung.
Aufgabe: Viele elementare Operationen parallel möglich.

1.2. Optisches Laufwerk

Land: Oberfläche, Pits: Vertiefungen.
Konstruktive bzw. Destruktive Interferrennz an den Vertiefungskanten.
EFM = Eight-to-Fourteen-Modulation. 14Bit statt 8Bit pro Byte nötig.
Multilayer: Halbtransparente und reflektierende Datenschicht.
Brenner: Nichtrefkletierende Schicht und unterschiedliche Intensitäten.

1.3. Festplatte

Permanent- und Elektromagnet für Schwenkarm.
Lesekopf Elektromagnet mit Ferritkern.
Informationsspeicherung durch Länge der Magnetfeldlinien.
Übereinanderliegende Spuren bilden einen Zylinder (parallele Schreib- und Lesevorgänge)

1.3.1 Ein Sektor 512 Byte Nutzdaten

60Byte Verwaltung: Sync des Taktes, Prüfsummen (CRC)
Nummer des aktuellen Zylinders/Sektors/Lesekopf.
Start und Endmarken für Datenbereich Leer-Byteäls Tolranzzone

1.4. Assemblerebene

Register: schneller Datenspeicher, welcher genau ein Wort abspeichern kann.
Es gibt allgemeine und spezielle Controll-Register `CR0, CR1, ...` Wort: Datenbreite die ein Prozessor auf einmal bearbeiten kann.
RAM: Speichert einzelne Bytes. Adressierung: Big Endian: speichert MSB, Little Endian: speichert LSB (Intel)
Anzahl der Operanden:
Dreiadressmaschine: $1 ← $2 + $3
Zweiadressmascine: $1 ← $1 + $2
Einadressmaschine: A ← $1 + A
Nulladressmascine: Stack-Maschine

Architekturen:
CISC: Complex Instruction Set Computer (veraltet)
kurze ,einfache Programme — lange Taktzyklen, kein Piplining.
RISC: Reduced Instruction Set Computer (vorherrschend)
einfache, gleich lange Befehle — mehr Speicher

2. Rechneraufbau

Peripherie – EAP – BUS – CPU – BUS – Speicher

Programmieren: Hochsprache(C++) → Assemblercode(ASM) → Maschinencode
Gesetz von Amdahl: Optimierung wirkt sich nur im Verhältnis der Häufigkeit eines Programmteils auf die Gesamtlaufzeit aus.
⇒ Optimiere die am häufigsten vorkommenden Programmteile.

3. MMIX Architektur

Hypothetischer μC von Donald Knuth für Lehrzwecke
Load-Store Architektur: Nur GPR mit ALU verbunden.
256 Register(32 SPR), 64Bit Wortbreite, 256 Befehle (1 Takt)
Big-Endian Adressierung

3.1. Register

`$0` – `rL-1`: lokale Register,
`rL` – `rG-1`: marginale Register(unentschieden)
`$255` – `rG`: globale Register

3.2. Hauptspeicher (RAM)

Text_Segment: 256Byte Interruptvektoren, Programmcode
Data_Segment: Für Daten(unten) und Stack(oben)
Pool_Segment, Stack_Segment, Betriebsystembereich

Speicherzugriffe: Byte(1), Wyde(2), Tetra(4), Octa(8)
Big-Endian: Adressen zeigen auf MSB (MMIX)
Little-Endian: Adressen zeigen auf LSB

3.3. Programmloader

Der Programmloader läd das Programm (führt den Befehl LOC aus) und such nach der `Main`-Marke bei der das Programm startet.

3.4. Befehle

Format: `Marke BEFEHL $1,$2,$3 Kommentar`
⇒ Max. 3 Leerzeichen pro Zeile, Operanden nur durch Komma getrennt.

3.4.1 Wichtige Befehle

`LOC Adresse`	Nachfolgende Befehle ab `Adresse` ausführen
`GREG Adresse`	Schreibt `Adresse` in Globales Register
`BYTE Wert`	Reserviert 1 Byte, initialisiert mit Wert
`SETL $X,YZ`	Setzt Bits 0–15 von `$X` auf `YZ`
`LD□ $X,$Y,($)Z`	Lade □ von Adr. `$Y+($)Z` in Reg. `$X`
`LDA $X,Label`	Lade Adr. von `Lable` in Reg. `$X`
`ST□ $X,$Y,($)Z`	Speichert Reg. `$X` in Adr. `$Y+($)Z`
`JMP Label`	Springt zu `Label`
`GO $0,Label`	Springt zu `Label` und speichert Folgeadr. in `$0`
`CMP $X,$Y,($)Z`	Vergleich; `$X = sgn($Y - ($)Z)`
`TRAP 0,Halt,0`	Beendet das Programm

LD□U: vorderen Bits werden mit Nullen gefüllt
LD□: vordere Bits werden mit Vorzeichen gefüllt
□=B(Byte/8b), W(Wyde/16b), T(Tetra/32b), O(Octa/64b)

3.4.2 Arithmetik

`ADD $X,$Y,($)Z`	`$X = $Y / $Z`
`SUB $X,$Y,($)Z`	`$X = $Y - $Z`
`MUL $X,$Y,($)Z`	`$X = $Y * $Z`
`DIV $X,$Y,($)Z`	`$X = $Y / $Z`

3.4.3 Verzweigung und Bedingungen

Branch	`B- $X,Label`	Springt zu `Label` if `$X`
Conditional Set	`CS- $X,$Y,($)Z`	`$X = $Z if $Y`
Zero or Set	`ZS- $X,$Y,($)Z`	`$X = $Z if $Y else $X = 0`

`-Z`	if Zero	`-NZ`	not zero
`-P`	if positive	`-NP`	not positiv
`-N`	if negative	`-NN`	not negative

Jeder Befehl hat 32 Bit:

OPC	#$1	#$2	Offset

Laden: RAM → Register, Speichern: Register → RAM

Namensraum: `PREFIX Prefix:` Vor alle Labels wird `Prefix:` geschrieben
Globale Labels mit : davor: `:Main`
Namensraum beenden: `PREFIX :`

3.5. ALU – Arithmetic Logical Unit

3.5.1 Schieber

Steuerung: 00,01,10,11 für SL,SLU,SR,SRU; nur SL liefert Überlauf!

3.5.2 Division

$a_1a_2a_3.../b_1b_2b_3... = c_1c_2c_3...$
Dividend Divisor Quotient
Wie bei Basis Zehn, nur dass Divisor maximal einmal abgezogen werden muss. Statt Divisor nach rechts schieben, Rest nach links schieben.

Ablauf: 1. Dividend in Restregister ablegen
2. Divisor in linker Hälfte des Divisorregisters ablegen.
3. Divisor = Divisor >> 1
4. Quotient = << 1
5. Rest = Rest − Divisor

6543/21

Restregister:	0000	6543
Divisorregister:	0021	0000

Restregister = Ergebnisregister (Rest links, Erg rechts)

`$X` $\mod n$: `DIV $X,$X,n` `GET $X,rR`
`if $Y ○ $Z: SUB buf,$Y,$Z` `B(N)P/N buf,Label`

3.6. Stack

Initialisieren des Stacks:

`BOS`	`GREG Pool_Segment`	Bottom-Of-Stack
`SP`	`GREG Pool_Segment`	Stackpointer

Bsp: Push und Pop von zwei Registern a und b:

	Push	Pop
1.	`SUB SP,SP,2*8`	`LDO b,SP,0`
2.	`STO b,SP,0*8`	`LDO a,SP,1*8`
3.	`STO a,SP,1*8`	`ADD SP,SP,2*8`

Funktionsaufruf der Funktion `Fkt`:

1. Variablen zur Übergabe auf den Stack speichern (Push)
2. Sprung zum Funktionscode: `GO $0,Fkt`
3. Von `Fkt` benötigte Register auf dem Stack sichern (Push)
4. Übergebene Variablen vom Stack in die Register laden (Pop)
5. Berechnungen in den gesicherten Registern durchführen
6. Übergabewert auf dem Stack mit Rückgabewert überschreiben (Push)
7. Alte Registerinhalte wiederherstellen (Pop)
8. Stackpointer auf die Adresse des Rückgabewerts setzen
9. Rücksprung `GO $0,$0,0`

4. Zustandsautomaten

Register: Haben immer aktuellen Wert am Ausgang, Änderung nur bei `clk`
`clk` Variablen mit steigender Taktflanke, müssen im vorherigen Zustand auf 0 gesetzt sein.
`Reset` Veriablen müssen meistens auf 0 gehalten werden.

4.1. Programmierbare logische Anordnung PLA

Eingänge → Zustände (AND) → Ausgänge (OR)

5. Datenpfade

Die 5 Phasen der Befehlsausführung:

1. BH: Befehl holen (IF = Instruction Fetch)
2. BD: Befehl dekodieren (ID = Instruction Decode)
3. AF: Befehl ausführen (EX = Execute)
4. SP: Speicherzugriff (MEM = Memory Access)
5. ES: Ergebnis schreiben (WB = Write Back)

5.1. Piplining

Trennung von sequentiellen Elementen durch Register. ⇒ Einzelelemente Parallelisierbar.
Datendurchsatz steigert sich max. um Faktor S: $D' = S \cdot D$
Max. Taktrate entspricht dem Kehrwert der Ausführungszeit der langsamsten Stufe: $f_{max} = \frac{1}{t_{max}}$

5.1.1 Datenkonflikte

Generell keine Konflikte falls ≥ 3 Befehle dazwischen!
Zwischen ES und BD: beim schreiben/lesen von $\$X$
Zwischen ES und AF: beim rechnen mit nicht gespeicherten Werten (kann durch Stall gelöst werden)
Zwischen `LD□ $X,*,*` und `ST□ $X,$Y,$Z`
Nicht zwischen `ST□ *,$Y,$Z` und `LD□ *,$Y,$Z`
Stalls nötig falls direkt nach einem `LD□`-Befehl das Zielregister `$X` für Berechnungen in der AF Phase benötigt wird.
Sonst kann der Konflikt durch einen Forward-Pfad gelöst werden.

5.1.2 Strukturkonflikt

Gleichzeitiges Lesen und Schreiben im Registerblock.
Tritt auf, falls 3 Befehle vor `ST□` das Register `$ X` gelesen wird.

5.1.3 Steuerungskonflikt

Direkt nach einer Registerneuzuweisung(`ADD $X,$Y,$Z`) wird ein durch dieses Register bedingte Operation ausgeführt(`BNZ $X,Label`). Dadurch ist nicht definiert welcher Befehl als nächstes kommt. Lösung: Befehlsumstellung,Stalls

6. Speicher

Register – Cache – Hauptspeicher – Festplatte

6.1. Cache

4MB:2^{22}bit

besteht aus s Sets mit jeweils r Rahmen mit jeweils B einzelnen Bytes.
Tag = Adresse der Daten im Speicher
Valid-Bit: Daten im Cache sind gültig

#Schlüsselbits = Bandbreite − #Setbits − #Bytebits

Direct Mapped Cache:

Fully-Associative-Cache:

Set-Associative-Cache:
$\#\text{Speicherbits} = \#\text{BAbits} + \frac{\#\text{Rahmenbits}}{\text{Set}} + \#\text{Setbits}$
Rahmengröße $= 2^B A$
Ein Rahmen pro Set

$\frac{\#\text{Zugriffe}}{\text{Zeit}}$ = Hit-Rate · Hit-Time + Miss-Rate · Miss-Time

Computersysteme 2
Apps – OS – Hardware
Betriebsystem: Grundsoftware zum ausführen, steuern und überwachen von Programmen.
System: Von der Umwelt mit Schnittstelle getrennte Menge an verknüpften Komponenten.

Begriffe:

busy waiting	Warten auf Ereignis mit ständiger Nachfrage
Mutal exclusion	Zugriffe auf gemeinsame Variable hintereinander
Race-Condition	Parallele Prozesse lesen und schreiben in gemeinsamer Variable.
SRPT	Shortest-Remaining-Process-Time (PV Strategie)
LCFS	Last Come First Served (PV Strategie:)
ELF	Executable and Linking Format (Ausführbare Dateiformat)
MMU	Memory Management Unit

7. Betriebsmittel BM

Klassifizierung:
Entziehbarkeit: unterbrechbar(UBM)/ununterbrechbar(DBM)
Wiederverwendbarkeit: einmalig(EBM)/wiederholbar(WBM)
Exklusivität: parallel benutzbar(PBM)/exklusiv(XBM)/teils(BBM)

Virtueller Speicher(4GB bei 32Bit): ca. 1GB für BS; 3GB für Code, Stack, Heap...

8. Prozesse

$\approx$ 100 Prozesse aus $\approx$ 1000 Threads laufen im Hintergrund von Windows/Linux.
User- und Systemmodus (mit previligierten Befehlen)
Zustände: running und ready
Verteilung der Rechenzeit an die Prozesse:
Scheduler: Verwaltet die Priorität von Prozessen. (Akt. neuen P)
Dispatcher: Schaltet Prozesse um. (Akt. Interrupts)

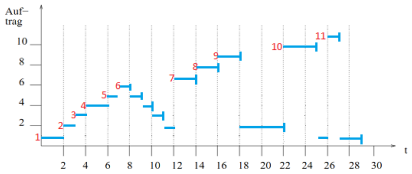
POSIX API		
Pförtner	Shell	
EA	Dateisystem	Prozessverwaltung
Unterbrechungen	Prozessschedule	
Hardware		

Jahr	Technologie	BS	Sonst
1945	Röhren	–	Ein Benutzer
1955	Transistoren	IBM	Lochkarten
1965	ICs	OS 360	Multitasking
1970	LSI	Unix	Dateisystem
1980	μC	Mirosoft	PC

UNIX-Scheduler: Round-Robin Zeitscheibenstrategie mit 100ms Quellcode
$\xrightarrow{\text{Compiler}}$ Binäre Objektdatei.o (Code + Daten)
Objektdatei*en* + Bibliothek*en* $\xrightarrow{\text{Linker}}$ Eine Executable
Executable $\xrightarrow{\text{Loader}}$ Prozess im RAM
ELF: Relocateable Files.o, Executable Files, Shared Object Files.so

8.1. Semaphore (Koordinationsvariable) m

Leser-Schreiber Problem: Es dürfen beliebig viele Leser auf eine Datei zugreifen solange niemand schreiben will.
Sobald ein Prozess in die Datei schreibt, werden neue Leser in die Warteschlagen gesetzt. Attribut, dass für beschränkte Ressourcen zur Zugriffskontrolle verwendet wird. Zunächst gilt $m = 1$, will jemand Datei beschreiben $m = m - 1$
falls $m = 0 \Rightarrow$ Zugriff, falls $m < 0 \Rightarrow$ warten
wenn Schreibvorgang fertig, dann $m = m + 1$



Auftrag	$A_i =$	1	2	3	4	5	6	7	8	9	10	11
Ankunftszeit	$a_i =$	0	2	3	4	6	7	12	14	16	22	26
Bedienzeit	$b_i =$	5	6	2	3	2	1	2	2	2	3	1

	$A_i =$	1	2	3	4	5	6	7	8	9	10	11
Abgangszeit	$c_i(LCFS,a,b) =$	29	22	11	10	9	8	14	16	18	25	27
Wartezeit	$w_i(LCFS,a,b) =$	24	14	6	3	1	0	0	0	0	0	0

- $LCFS$ (Last Come First Served):
  $\overline{W}(LCFS,a,b) = \frac{48}{11} \approx 4.4$ und $\overline{V}(LCFS,a,b) = \frac{77}{11} = 7$

8.2. Deadlocks (Verklemmung)

Def: Es gibt keinen Ablauf, so dass alle Forderungen nach Ressourcen erfüllt werden können.
Mind. zwei Prozesse warten gegenseitig auf die Freigabe der jeweils anderen Ressource.
Notwendige Bedingungen: BM ist exlusiv, BM is nicht entziehbar, Belegen und Anfordern, Zyklisches Warten
Lösungen: Ignore, Discover/Recovery, Avoidance (Verhinderung: Sorgfältige Zuweisung), Prevention (Vermeidung einer Bedingung)

Beispiel: Zwei Leute wollen etwas schreiben, der eine hat den Stift, der andere das Papier.
Wechselseitiger Ausschluss mit aktiven Warten kann zu Deadlocks führen, wenn der Prozess nicht unterbrochen werden kann.
Lösung: Passives Warten: Prozess wird vom Besetzer einer Ressource aktiviert wenn dieser sie nicht mehr benötigt.

8.3. Linux

Prozesse haben unterschiedliche Berechtigungen.
Priviligierte Aufrufe im Nicht-priviligierten Modus: systemcalls
Schutzkern: Fälschungssichere Objekt-IDs, Rechteverwaltung
Rechte: read (r, 4), write (w, 2) und execute (x, 1)
Für Besitzer, Gruppe und Andere (chmod 764, chown ...)
Datei: Folge von Bytes(regular, directories, devices(/dev)
Pipe: FIFO Komunikationspfad zwischen Programmen.
Links: Softlink(Symbol zB Pfadangabe) Hardlink(Zweiter INode)
Platte: SWAP | Inodes – Datenblöcke(cluster) – Reserveblöcke
Dateiverwaltung über INodes: ID, Typ, Rechte, Besitzer, Größe, Blöcke
Aufgabe Ausführung und Datenteile im Quellcode: Datenteile nur bei int,const, etc Befehlen!
Prozessstart:

1. Header Infos auf der ersten Seite lesen
2. Speicher allokieren
3. Blende Sectionen in Adressraum ein
4. Fehlende Biblitheken laden
5. Initialisiere Stack und Heap
6. Code ausführen

Verwaltungsstrategien LRPT/SRPT LPT/SPT LCFS/FCFS
Permutationsverfahren und zwei Zeitscheibenverfahren
Auftrag A_i, Ankunftszeitzeit a_i, Bedienzeit b_i, Abgangszeit c_i, Wartezeit w_i

$$\boxed{c_i - b_i = w_i + a_i}$$

$\overline{W} = \frac{1}{n}\sum w_i \quad \overline{V} = \frac{1}{n}\sum w_i + b_i$

9. Speicher

Häufigkeit der Nutzung: MFU, LFU
Zeitpunkte der Einlagerung: FIFO, LIFO
Zeitpunkt der letzten Nutzung: LRU, MRU

9.1. Virtueller Speicher VS

Gliederung: Segmente > Seiten > Zellen
Stretegie: First Fit, Rotating First Fit, Best Fit und Worst Fit
page fault (Seitenfehler): Seite nicht mehr im phys. RAM
VS: Bessere Nutzung kann real auch auf HDD liegen. MMU notwendig.
Jede Anwendung erhält virtuellen, linearen Speicherbereich
Seitenersetzung: FIFO, LIFO, LRU, LFU

Bei einem Speicher mit fünf Seitenrahmen, also mit $Frame_5 = \{f_1, f_2, f_3, f_4, f_5\}$, ergibt sich für die angegebenen Strategien folgendes Verhalten bezüglich der zu ersetzenden Strategien:

- **FIFO : First in First out:**

Referenzierte Seiten	f_1	f_2	f_3	f_4	f_5	Summe der Seitenfehler
1,3,5,4,2	1	3	5	4	2	5
4,3,2,1,0	0	3	5	4	2	6
5,3,5,0,4,3,5,4,3,2,1	0	1	5	4	2	7
3	0	1	3	4	2	8
4,5	0	1	3	5	2	9

Zur Erinnerung:
Hier wird erst f1 ersetzt, dann f2 usw., bis es wieder von vorne beginnt.

- **LIFO : Last in First out:**

Referenzierte Seiten	f_1	f_2	f_3	f_4	f_5	Summe der Seitenfehler
1,3,5,4,2	1	3	5	4	2	5
4,3,2,1,0	1	3	5	4	0	6
5,3,5,0,4,3,5,4,3,2	1	3	5	4	2	7
1,3,4,5	1	3	5	4	2	7

Hier wird quasi immer das letzte Element (f5) ersetzt, wenn eine Seite fehlt.

- **LRU : Least Recently used:**

Referenzierte Seiten	f_1,t	f_2,t	f_3,t	f_4,t	f_5,t	Summe der Seitenfehler
1,3,5,4,2	1,1	3,2	5,3	4,4	2,5	5
4,3,2,1,0	1,9	3,7	0,10	4,6	2,8	6
5	1,9	3,7	0,10	5,11	2,8	7
3,5,0,4	1,9	3,12	0,14	5,13	4,15	8
3,5,4	1,9	3,16	0,14	5,17	4,18	8
3,2	2,20	3,19	0,14	5,17	4,18	9
1	2,20	3,19	1,21	5,17	4,18	10
3,4,5	2,20	3,22	1,21	5,24	4,23	10

Zur Erinnerung:
Hier wird immer die Seite ersetzt, die am Wenigsten verwendet wurde (vgl. t).

- **LFU : Least Frequently used:**

Referenzierte Seiten	f_1,anz	f_2,anz	f_3,anz	f_4,anz	f_5,anz	$\sum$ Seitenfehler
1,3,5,4,2	1,1	3,1	5,1	4,1	2,1	5
4,3,2,1,0	1,2	3,2	0,1	4,2	2,2	6
5	1,2	3,2	5,1	4,2	2,2	7
3,5,0	0,1	3,3	5,2	4,2	2,2	8
4,3,5	0,1	3,4	5,3	4,3	2,2	8
4,3,2,1	1,1	3,5	5,3	4,4	2,3	9
3,4,5	1,1	3,6	5,4	4,5	2,3	9

9.2. Datenbanken Systeme (DBS)

Entity Relationship Modell (ER-M):
Rechteck: Objekte, Ellipse: Attribute, Raute: Methoden, Beziehungen
Schlüssel: Minimale Menge von Attributen, die ein Entity(Objekt) eindeutig in der Menge seiner Entities identifiziert (z.B. MatNR)

Datenbankentwurf:

- Existenzabhängigkeit von Entities (schwache und starke Entities)
- Generalisierung und Spezialisierung
- $is-a$ und $part-of$ Beziehungen
- Verschiedene Sichten (Rechte, Zugriffe)

Das relationale Modell Mengenorientierte Verarbeitung

9.3. SQL Befehle

`SELECT` * bzw. Attribut1, Attribut2,... (werden ausgegeben)
`FROM` Objekt1, Objekt2, ... (alle Benötigten Quellen auch von WHERE)
`WHERE` Bedingung1 `AND/OR` Bedingung2
`ORDER BY` Attribut1 (`DESC`), Atrribut2 **;**

Bedingung: (`NOT`) Attribut =, <, > Zahl/"Wort"
`SELECT DISTINCT` Attribut: Listet Einträge von Attribut max. einmal
`WHERE` Name `LIKE` "Mar%" bzw. "%tin": Alle Namen die mit "Ma" beginnen bzw. mit "tin" enden.
Equi-Join Bedingung: Objekt1.Atrribut = Objekt2.Attribut
Verständnis: `FROM` bildet alle mögl. Tupel der Objekte

Normalform:
N1: Zusammengesetzte Werte müssen getrennt werden und Wiederholungen entfernt werden
N2: Jedes nicht-primäre Attribut ist vom gesamten Primärschlüssel abhängig
N3: Ein Attribut, dass nicht zum Schlüssel gehört, darf nicht von anderen Nicht-Schluessel-Attributen abhängig sein