Themengebiete „Rechnerarchitektur“

- **Grundlagen**
  - Rechner
  - Interne/Externe Architektur
  - Moore’s Law
  - Design/Verifikation Gap
  - Entwurfsprinzipien
- **Abstraktionsebenen**
  - Systemebene
  - Algorithmische Ebene
  - Register Transfer Ebene
  - Gattereben Transistorebene
  - Layoutebene
  - Y-Diagramm
- **Hardwarebeschreibungssprachen**
  - Softwareprogrammiersprache vs. HDL
  - Verilog
- **Hardwaresimulation**
  - Simulationstechnike
    - Streamline Code Simulation
    - Critical Event Scheduling
  - Verifikation durch Simulation
- **Formale Verifikation**
  - Aquivalenzprüfung
  - Modellprüfung
  - Theorembeweis
- **Timinganalyse**
  - Totzeit
  - Totzeiten für steigende und fallende Flanken
  - Träge Totzeit
  - Kritischer Pfad
- **Register-Transfer-Ebene**
  - Ablaufplanung
  - Datenabhängigkeiten
  - ASAP, ALAP
  - Lebenszeit von Variablen
  - Synthese des Steuerwerks
- **Gatterebene**
  - Gattersynthese
    - PLA
    - Bibliothekselemente
    - FPGA
    - Standardzellen
  - Partitionierung
    - Kernighan-Lin
    - Simulated Anealing
    - Slicing
  - Verdrahtung nach Lee
- **Zahlendarstellungen**
  - Logisches und arithmetisches Schieben
    - Schieberegister
  - Integers
    - B&V
    - Einerkomplement
    - Zweierkomplement
  - Addierer
    - Überlauferkennung
    - Subtrahieren mit Addierer
  - Eine einfache ALU
  - Multiplizierer

        - Zweierkomplementzahlen nach der Schulmethode
        - Verfahren von Booth
    - Dividieren
        - Division von Zweierkomplementzahlen nach der Schulmethode
        - Abgewandelte Schulmethode (mit Restore)
    - Carryoverflowbehandlung
        - Wrap-Around vs Sättigungsarithmetik
    - Fließkommazahlen
        - Nachteile von Festkommazahlen
        - IEEE754
        - Normalisierte Zahlen
        - Bias
        - NULL und unendlich
        - Gültigkeit von Rechengesetzen
        - Addition von IEE754
        - Multiplikation von IEEE754
- **Systemebene**
    - Von Neumann Rechner
        - Bustechnologie
        - CPU
        - Hauptspeicher
            - Alignierter Speicher
                - Adressierung
        - Registerspeicher
            - Homogene vs. Inhomogene
            - Spezialregister
- Befehlsschnittstelle
    - Befehlsklassen
        - Transportbefehle
        - Arithmetische Befehle
        - Logische Befehle
        - Bitverarbeitende Befehle
        - Schiebe- und Rotationsbefehle
        - Stringbefehle
        - Sprungbefehle
        - Systembefehle
        - Priviligierte Befehle
        - Trap-Befehle
    - Speicheradressierung
        - 0-Stufig, 1-Stufig, 2-Stufig, Mehrstufig
        - 3-Adressmaschine
        - 2-Adressmaschine
    - n-Adressmaschinen
        - 1 ½ Adressmaschine
        - 1-Adressmaschine
        - 0-Adressmaschine (Stack)
    - Befehlstypen
        - Register-Register
        - Register-Speicher
        - Speicher-Register
        - Speicher-Speicher
    - Mikroprozessoren
        - RISC vs. CISC
        - Exceptions
        - Interrupts
            - Maskierte und unmaskierte Interrupts
            - Interrupt Service Routine (ISR)
            -
        - ALU
            - Maschinenbefehle
        - Pipelining

- Beispiel
- Vorraussetzungen
- Speedup
- Registerbänke
- Datenabhängikeiten
    - True-Dependence
    - Anti-Dependence
    - Output-Dependence
- Aufgabe des Compilers beim Pipelining
    - Hazards
- Techniken
    - Result Forwarding
    - Registerbypassing
- Kontrollhazards
    - Software-/Hardwaremäßige Lösung
    - Delayed Branch Technik
    - Statische vs. Dynamische Sprungvorhersage
        - Entscheidungsautomaten
- Funktionale Einheiten
    - Scheduling
        - Scoreboarding
        - Registerumbenennung
        - Tomasula Algorithmus
            - Reservation Stations
            - Common Data Bus
            - 3-Phasen des Tomasulo Algorithmus
    - Superskalarprocessing
        - Zuordnungsbandbreite
        - Superskalarpipeline
        - Issue, Dispatch, Completion und Commitment, Retirement
        - Superskalar vs. VLIW

- Aufbau eines Rechners
    - Speicherhierarchien
        - Festwertspeicher
        - S-RAM vs. D-RAM
        - Cache
            - Cache-Hit vs- Cache Miss
            - Zugriffszeiten mit Cache
            - Verdrängungstrategien
                - LRU
                - LFU
                - FIFO
            - Aufbau
                - Adresspeicher, Datenspeicher
            - Vollassoziativer Cache Speicher
            - Direct Mapped Cache Speicher
            - Satzassoziativer Cache
            - Ursachen für Cache Misses
            - 3 Varianten für Schreibzugriffe
                - Write-Through
                - Write-Back
                - Write-Allocation
            - Early Restart und Out of Order fetch
            - 2^{nd} Level Cache
        - Virtueller Hauptspeicher
            - Paging
            - Kenndaten
            - Translation Lookaside Buffer
            - Seitenersetzungstrategien

- LFU
- FIFO
  - FIFO-Variante (2nd Chance)
- LRU
- NRU

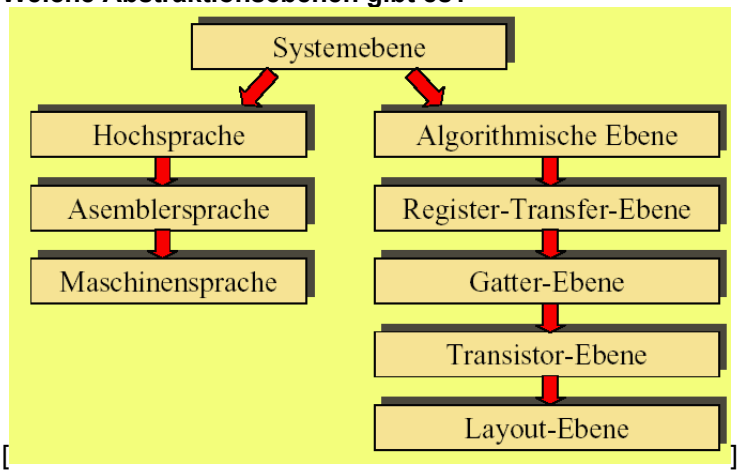
- Hauptspeicher
  - Segmentierung
  - Paging und Segmentierung kombiniert
  - Physikalische und virtuelle Cacheadressierung
- Festplatte
  - Magnetismus
  - Schreiben
  - Lesen
  - Schreibleseköpfe
  - Kodierung
  - Festplattenzugriff
    - FCSF (First-Come-First-Served
    - SSF (Shortest Seek First)
    - Fahrstuhlalgorithmus
  - Formatierung
  - Kontinuierliche Allokation vs. Allokation mit verketteter Liste
  - FAT, I-Node
  - Asynchrone vs. Synchrone Kommunikation
- Interface
  - Port
  - Bus
  - Interfaceprotokoll
    - Lesen und Schreiben
      - Strobe Protokoll
      - Handshake Protokoll
      - Mixed
  - E/A Adressierung
    - Port-Based
    - Bus-Based
  - Busse
    - Priority Arbitrierung
    - Daisy-Chain Arbitrierung
    - Daisy-Chain Arbitrierung mit Priority und Round Robin
  - Kommunikationstechniken
    - Programmierte E/A
    - Interrupt
    - DMA
    - Memory Mapped
    - I/O Mapped
  - Synchronisierung
    - Strobing
    - Polling
    - Handshaking
  - DMA Zugriff
    - Cycle Stealing
    - Memory Idle

- **Parallelrechner**
  - SISD, SIMD, MISD, MIMD
  - Charakteristiken
  - Statische vs. Dynamische Netze
    - Hypercube
      - eCube Routing

        - Cube Connected Cycle
        - Crossbar-Switch
    - Permutationsnetze
        - Typen
            - Perfect-Shuffle
            - Kreuzpermutation
            - Tauschpermutation
            - Umkehrpermutation
            - Omega Netzwerk
            - Butterfly
            - Banyan
            - Delta
            - Benes
        - Self Routing Property
        - Deterministische vs. Adaptive Wegewahl
        - Phit
        - Store and Foreward Modus
        - Virtual Cut Through
    - Leistungsbewertungen bei Parallelrechnern
        - MIPS
        - Benchmarks
        - Speedup
            - Amdahles Gesetz
    - Cache-Kohärenzprotokolle
        - Snooping Protokolle
            - Write-Once
            - Firefly
            - Pentium MESI
        - Directory-Based Protokolle
        - Softwarebasierte Verfahren

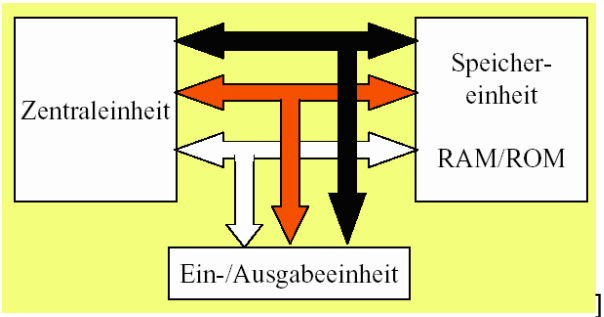
Fragenkatalog Rechnerarchitektur:

1. **Was ist ein Rechner?** [„Ein Rechner ist ein universell einsetzbares Gerät zur automatischen Verarbeitung von Daten“ - Duden]
2. **Was versteht man unter externer bzw. interner Architektur?** [externe Architektur: Der Begriff architektur wird hier benutzt um die Teiles eines Systems zu beschreiben, so wie es der Programmierer/Entwicklur sieht
   interne Architektur: Bezeichnet die Organisation und interconnection(Verbindung) von Komponenten eines Computer Systems]
3. **Was besagt Moore’s Law?** [Verdopplung der Transistor-Dichte alle 18 Monate]
4. **Kennen Sie Rückschläge der Computerentwicklung?** [Pathfinder, Challenger Unglück, Pentium Bug]
5. **Was versteht man unter dem Designgap?** [Die Größe von Systemen übersteigt die Fähigkeiten im Entwurf auf niedriegen Abstraktionsebenen]
6. **Was ist der Design-/Verifikationgap?** [Die Fähigkeit ein großes System zu Designen, steigt nicht in dem Maße an, wie der Bedarf nach neuen Entwicklungen. Die Fähigkeit zu verifizieren steigt noch langsamer, der Verifikationgap also noch größer als der Designgap]
7. **Was versteht man unter Top-Down bzw. Bottom-Up Entwurf?** [Top-Down: Abstrakte Beschreibung des Systems wird sukzessive verfeinert bis zur Hardware |
   Bottom-Up: Bereits entworfene Komponenten werden zu komplexen Einheiten zusammengefügt]
8. **Welche Abstraktionsebenen gibt es?**



[ ]

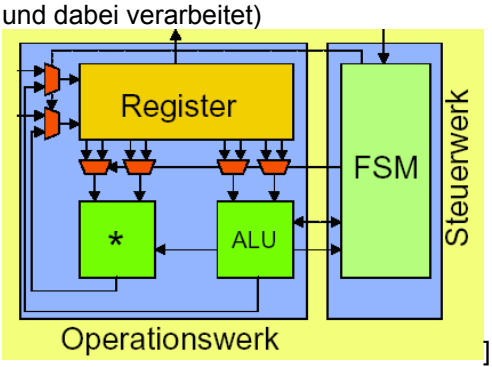
   a. **Was ist die Systemebene?** [Darunter versteht man die Einheiten die miteinander kommunizieren:



]

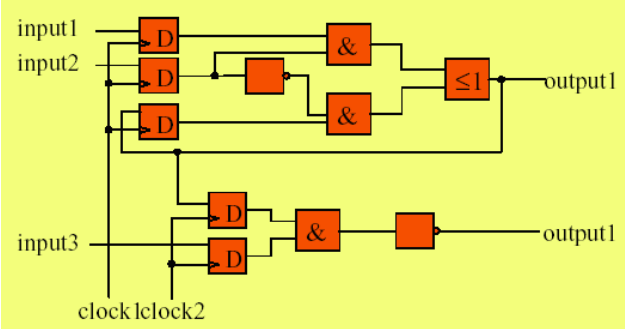
   b. **Was ist die Algorithmische Ebene?** [Die Funktionen einzelner Blöcke wird durch Algorithmen in einer HW Beschreibungssprache spezifiziert]
   c. **Was ist die Register Transfer Ebene?** [Darstellung der Funktionalen Einheiten durch Datenpfad und Kontrollpfad (Daten werden von Register zu Register transferiert

und dabei verarbeitet)

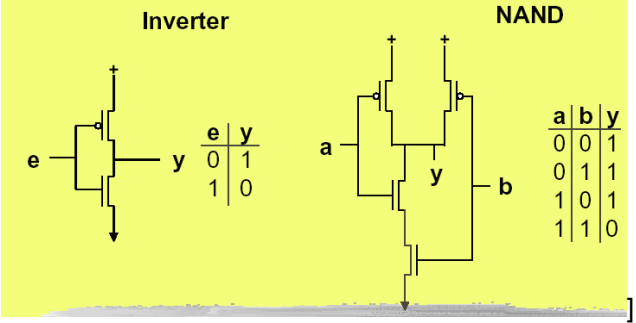


]

d. **Was ist die Gatterebene?** [Es gibt nur noch boolesche Signale, boolesche Gatter und einfache Flip-Flops

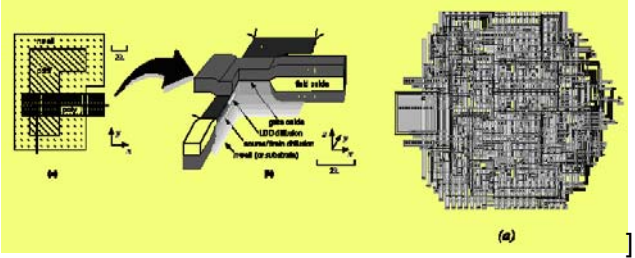


e. **Was ist die Transistorebene?** [Realisierung boolescher Elemente durch Transistoren



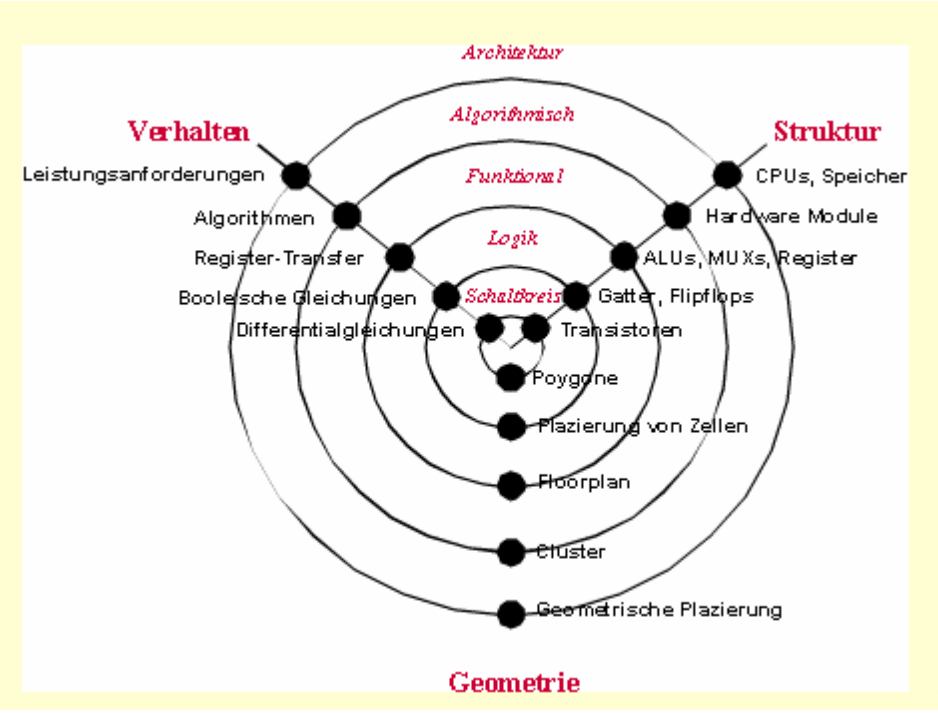
]

f. **Was ist die Layoutebene?** [Realisierung von Transistoren durch dotierte Bereiche und isolierende Schichten auf dem IC



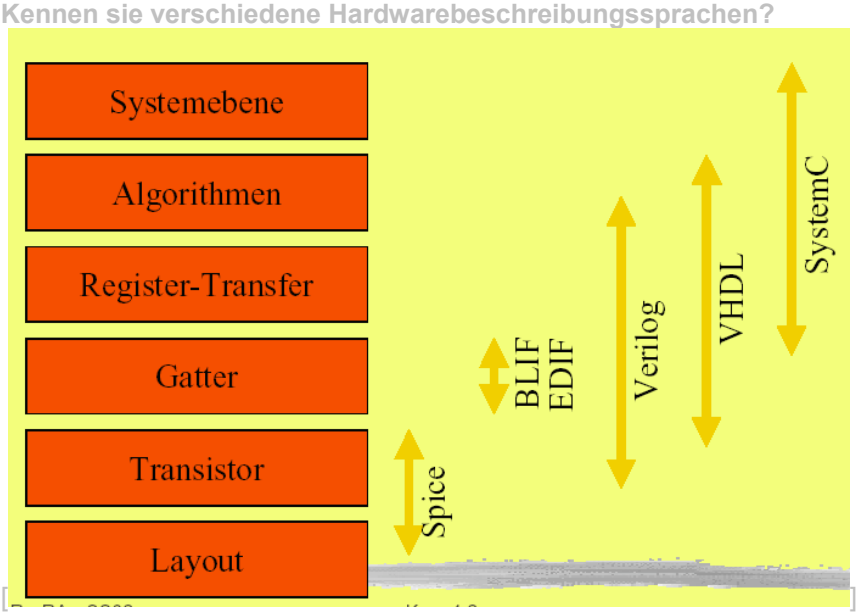
]

9. **Was wird im Y-Diagramm dargestellt?** [Struktur/Verhalten/Geometrie der einzelnen Abstraktionsebenen



]

10. **Welche Eigenschaften sollte eine Hardwarebeschreibungssprache (HDL) mitbringen?** [präz. Spezifikation/Simulation/ Automatisierung/ Dokumentation/ Beschleunigung]
11. **Kennen sie verschiedene Hardwarebeschreibungssprachen?**

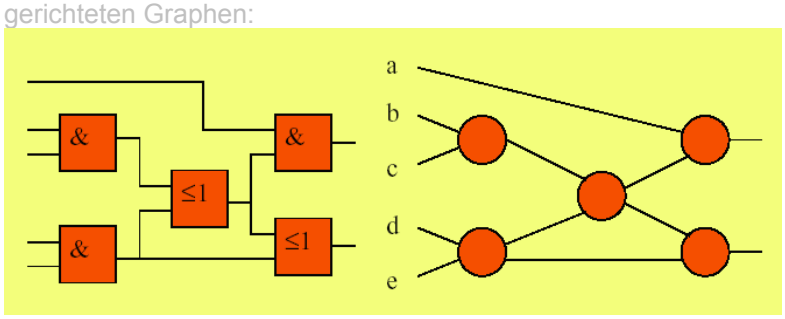


]

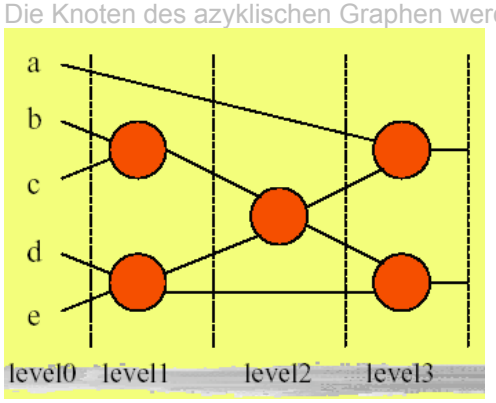
12. **Was ist der Unterschied zwischen eine Softwareprogrammiersprache und einer HDL?** [In einer HDL werden Schaltungen beschrieben und diese zu Hierarchien zusammengefasst]
13. **Weshalb wird eine Algorithmische Beschreibung eher angewandt als eine strulturelle Beschreibung?** [Strukturelle Beschreibung ist zu komplex für große Entwurfe mit z.B. 20 Millionen Gattern]
14. **Welche 4 Besonderheiten hat Hardware dabei?** [Zeitbegriff➔ Funktionen verbrauchen Zeit/parallele Tasks➔Funktionen können parallel arbeiten | Signale und Ereignisse ➔ Kommunikation zwischen Modulen | mehrwertige Logik(0,1,x,z…),➔ Zweiwertige Logik reicht nicht aus]
15. **Was genau ist Verilog? Welche Abstraktionsebenen deckt dieses ab?** [Verilog ist ein Standard mit einheitlichen Schnittstellen zwischen Werkzeugen und Firmen | Es ist portabel, d.h Entwürfe können durch verschiedene Synthesewerkzeuge optimiert und durch

verschieden Analysewerkzeuge simuliert werden ➔ Technologie und Firmenunabhängig, d.h. man kann den Technologiepartner ohne weiteres wechseln]

16. Wie übertragt man ein Gatter bzw. eine kleine Schaltung von Gattern in ein Verilog Programm, z.B. MUX? []
17. Im Gegensatz zu einer normalen Programmiersprache gibt es bei Verilog „Parallele Schachtelung“. Wie werden diese definiert?
18. Was sind Ereignisse?
19. Was versteht man unter Levelabhängigem Warten?
20. Welche Arten von Schleifen gibt es bei Verilog?
21. Was ist Procedural assignment, was Continous Assignment?
22. Wozu dienen Tasks?
23. Was sind Funktionen?
24. Was sind parametrisierte Module?
25. Wie kann man eine Instanz erzeugen?
26. Welche Systemtasks gibt es?
27. **Welche Aufgabe hat ein Simulationsalgorithmus?** [Abbilden der Hardwarefunktionalität auf eine Zielarchitektur wie z.B. den von Neumann Rechner – Ausgangspunkt ist z.B. die Hardwarebeschreibung in Verilog o.A. – Problem: Hoher Grad an parallelität muß auf eine sequentielle Maschine projeziert werden]
28. **Welche Arten der Hardwaresimulation gibt es?** [Analog: Wert und Zeitkontinuierlich, nichtlineare Differentialgleichungen ➔ Spice |Digital: wert- und Zeitdiskret, boolesche Algebra ➔ VSS, modelsim | Mixed: analog/digital-simulation ➔ Saber]
29. **Welche Simulationstechniken gibt es?** [Streamline Code Simulation➔Schaltnetze, vollsynchrone Schaltwerke | Equitemporal Iteration: ➔ Analogsimulation | Critical Event Scheduling: Schaltungen auf unterschiedlichen Abstraktionsebenen und mit Zeitinformationen]
    a. **Beschreiben Sie die Streamline Code Simulation genauer!** [Man nennt diese auch Comiled Mode, es wird direkt ausführbarer Code auf der Zielpalttform erzeugt. Einschränkungen ergeben sich auf kombinatorische oder strikt synchrone Schaltungen. Keine Zeitinformation. Man notiert die Schaltung als azyklischen gerichteten Graphen:



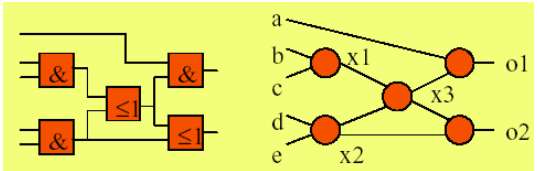
Die Knoten des azyklischen Graphen werden halbgeordnet (levelizing).



Knoten auf einer höheren Ebene können Knoten auf niedrigeren Ebenen nicht beeinflussen – Knoten auf der selben Ebene beeinflussen sich nicht, ABER Knoten auf niedrigeren Ebenen beeinflussen Knoten auf höhern Ebenen.]

        i. **Wie wird der Code generiert?** [Die Level werden nacheinander implementiert. Die Reihenfolge der Operationen in den Levels ist beliebig. Gatter werden durch Operatoren der Zielmaschine realisiert. Die

Verbindungen (Netze) werden durch Variablen der Zielmaschine implementiert]



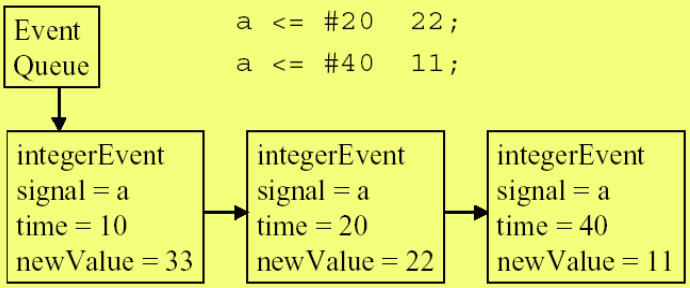

```
int a, b, c, d, e;
int x1, x2, x3;
int o1, o2;

x1 = b & c;     // level 1
x2 = d & e;     // level 1
x3 = x1 | x2;   // level 2
o1 = a & x3;    // level 3
o2 = x2 & x3;   // level 3
```

ii. **Wie sieht das aus?** [

b. **Wie funktioniert Critical Event Scheduling?** [
Die Idee ist es algorithmische Blöcke durch Threads zu verwalten. Nur die Systemteile werden neu berechnet, dern Eingaben sich verändert haben ➔ Events.
Es gibt eine globale Datenstruktur, die Event-Queue
Ein spezieller Thread (Scheduler) übernimmt die Verwaltung]

i. **Was sind Ereignisse?** [Änderungen eines Signalwertes oder von Zeitoperatoren generierte Events. Sie treten zu einem festen Zeitpunkt auf. Der Simulationsalgorithmus speichert eine sortierte Liste der anstehenden Ereignisse

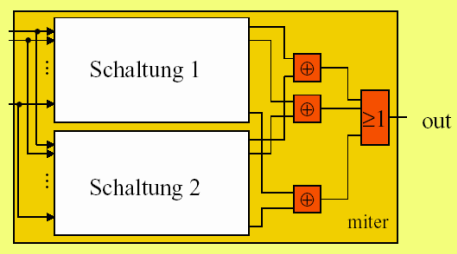
```
a <= #10  33;
a <= #20  22;
a <= #40  11;
```



Um bei Eintreten von Ereignissen die Simulation weitertreiben zu können, müssen die Threads gespeichert werden (zu jedem Event), die gerade auf Events warten.]

ii. **Durch diese ereignisbasierte Simulationstechnik ergeben sich zwei Zeitachsen, welche?** [Simulationszeit | Delatzeit: Enstehehen dadurch, dass mehrere Ereignisse zur gleichen Simulationszeit ablaufen
]

30. **Was geschieht bei der Verifikation durch Simulation?** [Eingangsstimuli erzeugen/Ausgangssignale beobachten]
31. **Welche 3 Formen der formalen Verifikation gibt es?** [Äquivalenzprüfung/Modellprüfung/Theorembeweis]
32. **Wie funktioniert die Äquivalenzprüfung bei kombinatorischen bzw. sequentiellen Schaltungen?** [Die Frage ist, ob zwei gegebene digitale Schaltungen die gleiche Funktionalität haben. Bei kombinatorischen vergleichen wir die Ausgangssignale bei gleichen Eingangssignalen bei beiden Schaltungen | Bei sequentiellen Schaltungen: Sind die Ausgänge zu allen Zeitpunkten bei gleichen Eingabefolgen identisch. Das kann man überprüfen mit einem „miter“:

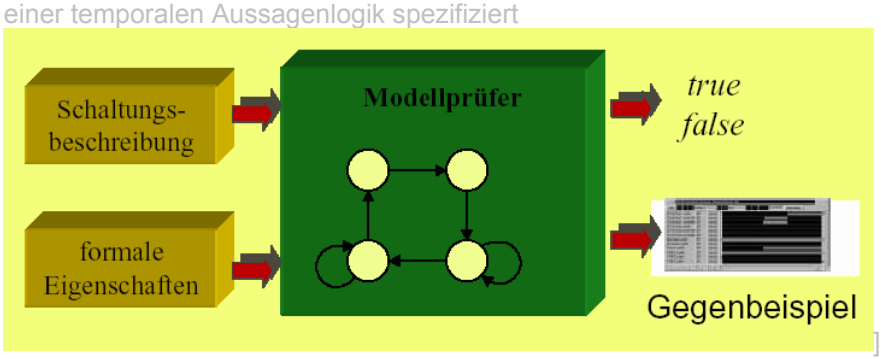


Liefert der Miter „0“ sind die Schaltungen für diese Eingangsbelegung äquivalent]

a. **Wie funktioniert die Äquivalenzprüfung bei Schaltnetzen?** [Zunächst transformieren wir das Schaltnetz in einen Normalformdarstellung (z.B. KNF, ROBDD,…)➔ Liegt Äquivalenz der Normalformen vor? ➔ Optimierungsmöglichkeiten.]

b. **Wie funktioniert die Äquivalenzprüfung bei Schaltwerken?** [Schaltungen in Huffman-Normalform bringen, wenn gleiche Zustandskodierung und ein eindeutiger Resetzustand vorliegen, dann kann man das Problem auf die Äquivalenzprüfung von Schaltnetzen zurückführen. Hat man keine gleiche Zustandskodierung oder gemeinsame Rücksetzleitung, kann man das Problem mit Hilfe der Automatentheorie lösen und den Miter in Form eines Produktautomaten bilden. Man durchsucht den Zustandsraum nach Zuständen, die true am Ausgang erzeugen]

33. **Was ist Modellprüfung?** [Die Frage ob ein Schaltwerk eine gegebene Spezifikation erfüllt. Das Schaltwerk wird durch endliche Automaten modelliert. Die Eigenschaften werden in einer temporalen Aussagenlogik spezifiziert



]

a. **Warum benötigt man hierzu die temporale Aussagenlogik?** [Um zeitliches Verhalten auszudrücken]

b. **Was ist denn Temporallogik (LTL)?** [Die Temporallogik besteht aus atomaren Ausdrücken➔Signale des Designs und Boolesche Funktionen. Es gibt die üblichen booleschen Operatoren (wie bei der Aussagenlogik) And, Or, NOT,… . Desweiteren gibt es nun temporale Operatoren

$X_{[m]}\ \varphi$	in genau m Zeitschritten gilt φ
$F_{[m,n]}\ \varphi$	im Intervall [m,n] gilt φ mindestens einmal
$G_{[m,n]}\ \varphi$	im Intervall [m,n] gilt immer φ
$\varphi\ U_{[m,n]}\ \psi$	ψ wird im Intervall [m,n] wahr und bis dahin gilt immer φ

]

i. **Welche typischen Eigenschaften besitzt die LTL?** [

Sicherheit: a und b werden nie gleichzeitig war: $G\ \neg(a \wedge b)$

Lebendigkeit: Jede Anforderung wird innerhalb von 5 bis 7 bestötigt:

$G\ (req \rightarrow F_{[5,7]} ack)$

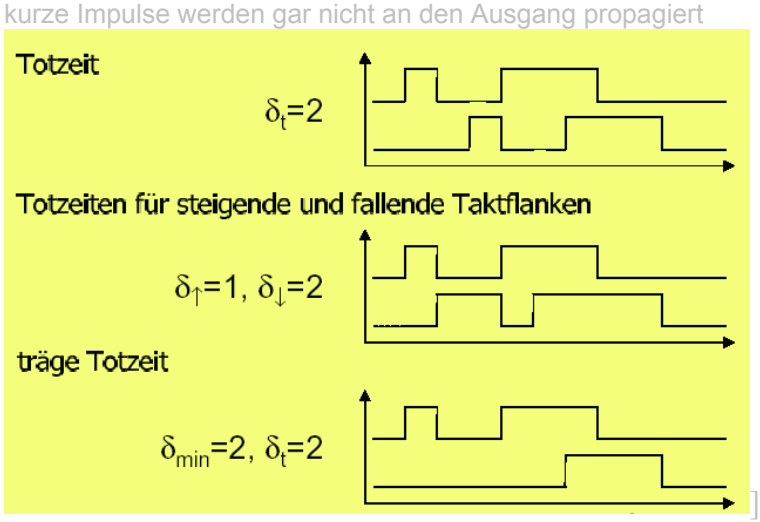
Fairness: Innerhalb von 30 Schritten gilt mindestens einmal a: $G\ F_{[30]} a$]

ii. **Wie ist die semantik der Temporallogik?** [Man hat eine Menge von Signalen und ein Abbildung π, die jedem Signal einen Zeitpunkt zuordnet.

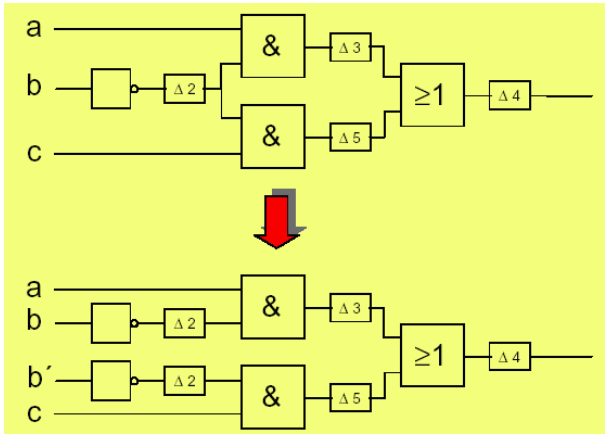
$$\pi \models a \quad \Leftrightarrow \quad \pi(a,0)=true$$
$$\pi \models \neg\varphi \quad \Leftrightarrow \quad \pi \not\models \varphi$$
$$\pi \models \varphi \wedge \psi \quad \Leftrightarrow \quad \pi \models \varphi \text{ und } \pi \models \psi$$
$$\pi \models X_{[m]}\,\varphi \quad \Leftrightarrow \quad \pi^m \models \varphi$$
$$\pi \models F_{[m,n]}\,\varphi \quad \Leftrightarrow \quad \exists\, m \le t \le n \,.\, \pi^t \models \varphi$$
$$\pi \models G_{[m,n]}\,\varphi \quad \Leftrightarrow \quad \forall\, m \le t \le n \,.\, \pi^t \models \varphi$$
$$\pi \models \varphi\, U_{[m,n]}\,\psi \quad \Leftrightarrow \quad \exists\, m \le t \le n \,.\, \pi^t \models \psi \text{ und } \forall\, k < t \,.\, \pi^k \models \varphi$$

iii. **Warum ist es nicht sinnvoll die Korrektheit einer Schaltung über Signalabfolgen zu testen?** [Weil es davon unendlich viele gibt, das Problem dadurch semi.entscheidbar wird, deshalb löst man das an einem formalen Modell, einem endlichen Automaten]

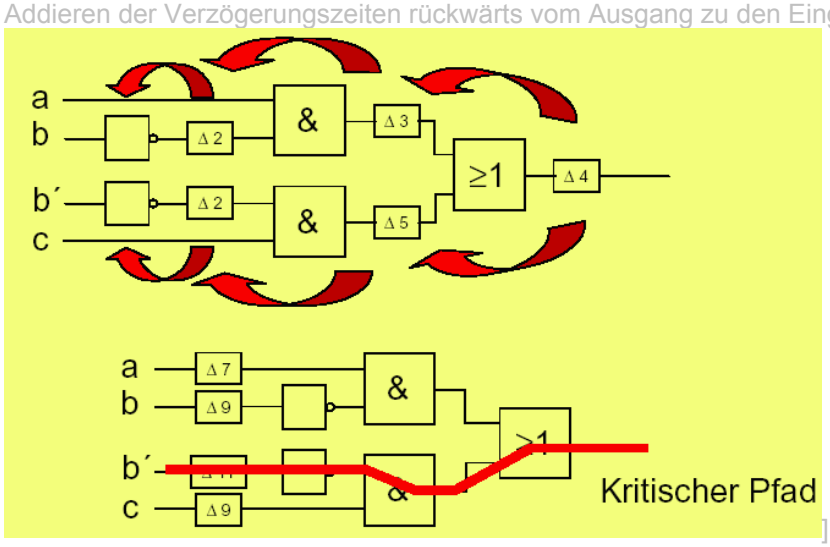
34. **Was ist Theorembeweisen?** [Man modelliert die Schaltung und deren Eigenschaften in einer Logik höherer Ordnung und beweist die interaktiv durch die jeweiligen Theorem der Logik. Kombinieren kann man dies mit diversen Heuristiken zur Automation des Beweises und durch automatische Entscheidungsprozeduren]
35. **Was wird bei der Timinganalyse bestimmt?** [Wie hoch man die Schaltung takten kann]
    a. **Welche Zeitmodelle für Gatter gibt es?** [
    Totzeit: Das Eingangssignal erscheint zeitverschoben am Ausgang |Totzeiten für steigende und fallende Taktflanken sind unterschiedlich | Die Totzeit ist träge, d.h. kurze Impulse werden gar nicht an den Ausgang propagiert



    ]
    b. **Wie könnte man die Totzeiten einfach in ein Modell packen?** [1. Möglichkeit: Man vernachlässigt die Totzeiten | 2. Möglichkeit: Man ordnet jedem Gatter ein Verzögerungsglied zu.]
    c. **Welche beiden Schritte muss man bei der Timinganalyse beachten?** [Duplizieren von mehrfach benutzten Schaltungsteilen und Eingängen:



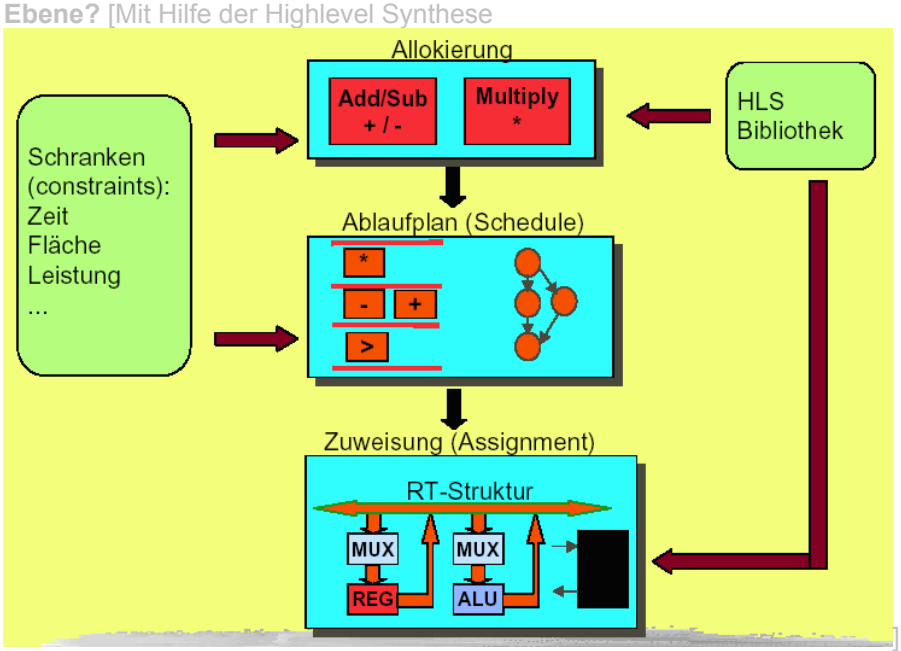
Addieren der Verzögerungszeiten rückwärts vom Ausgang zu den Eingängen hin:



]

d. **Was beudet ob der längste Pfad (kritischer Pfad) auch sensibilisierbar ist?** [d.h. ob es einen Eingangssignalwechsel gibt, der sich über den längsten Pfad zum Ausgang durchsetzt➔ D-Kalkül (nicht erklärt worden)]

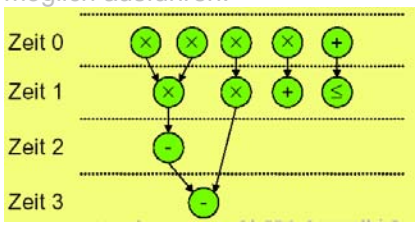
36. **Wie kommt man vom Hardwarebeschreibenden Algorithmus zur Register-Transfer-Ebene?** [Mit Hilfe der Highlevel Synthese



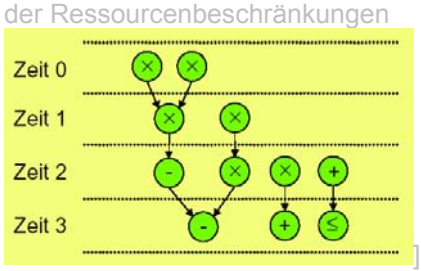
]

a. **Was versteht man unter Allokierung?** [Abbildung von Operatoren der Programmiersprache in HW-Komponenten ➔ Die Komponenten liegen in Bibliotheken vor ➔ ALUs, Multiplizierer, etc. jeweils mit generischer Bitbreite]

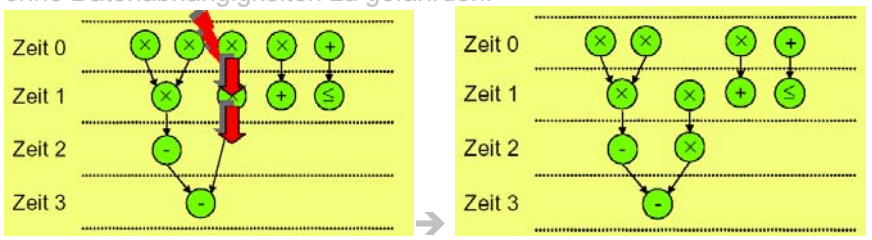
b. **Was ist Ablaufplanung?** [Bestimmung der Zeitpunkte, zu denen die Operationen auszuführen sind unter Berücksichtigung von Ressourcenbeschränkungen (z.B. nur ein Multiplizierer) und Zeitbeschränkungen (maximale Latenz)]
c. **Wie reagiert man auf Datenabhängigkeiten?** [Wir ordnen den Knoten im Problemgraph Ausführungszeitpunkte zu (Feed-Forward Graph)]
d. **Was beschreiben die Strategien ASAP und ALAP?** [Mit den beiden Strategien kann man die Ressourcenbeschränkungen einhalten. ASAP: Operationen so früh wie möglich ausführen.

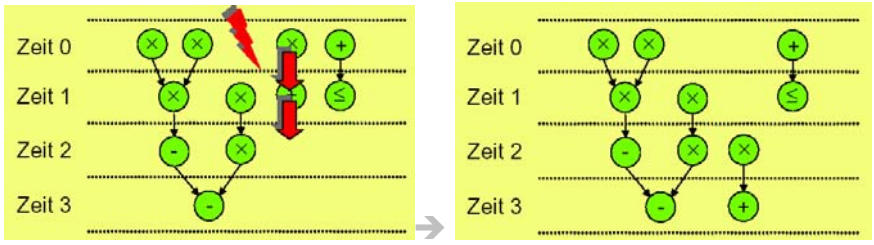


ALAP: Operationen so spät wie möglich ausführen. Beide Strategien unter Einhaltung der Ressourcenbeschränkungen



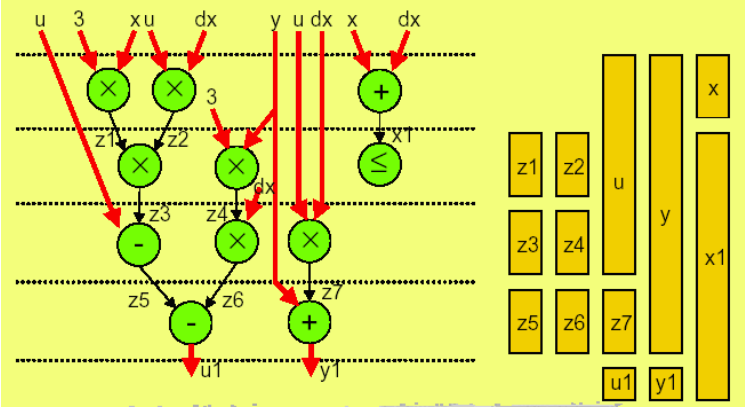
]
e. **Wie reagiert die ASAP Strategie, wenn es eine Beschränkung auf z.B. 2 Multiplizierer und 2 ALUs gibt?** [Die Operationen müssen im Ablaufplan gemäß der ASAP Strategie verschoben werden um die Ressourcenabhängigkeiten einzuhalten ohne Datenabhängigkeiten zu gefährden.





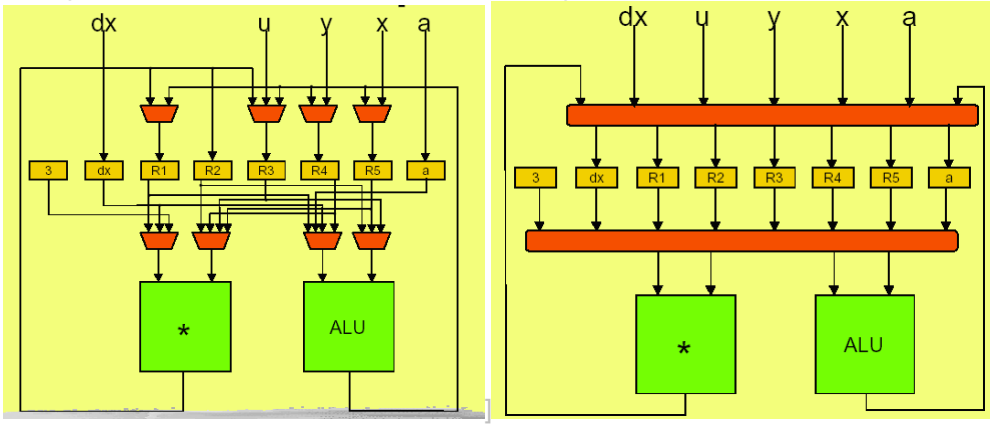
Das Problem ist für kombinierte Zeit- und Ressourcenbeschränkungen NP-vollständigkeit]
f. **Welche weiteren Algorithmus zur Lösung dieses Problems gibt es noch?** [Listenscheduling, Force-direct-scheduling | Ganzzahlige lineare Programmierung]
g. **Was versteht man unter der Bindung von Registern und der Lebenszeit einer Variablen?** [Zuordnung von Ressourcen zu Operationen. Dies kann vor, während oder nach der Ablaufplanung durchführen. Bindung nach der Ablaufplanung ist in polynomieller Zeit möglich. Zuordnung von Speicherstellen/Registern zu

Programmvariablen gehört auch zur Bindung!

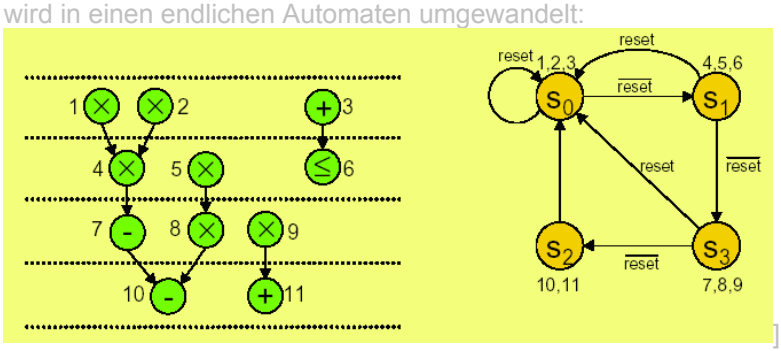


37. **Welche Vor- und Nachtteile haben Busse bzw. MUXe bei der Synthese des Operationswerkes?** [Multiplexer: Hoher Verdrahtungsaufwand, aber dafür paralleler Datentransport Busse: Geringer Verdrahtungsaufwand, aber dafür eventuell Flaschenhals

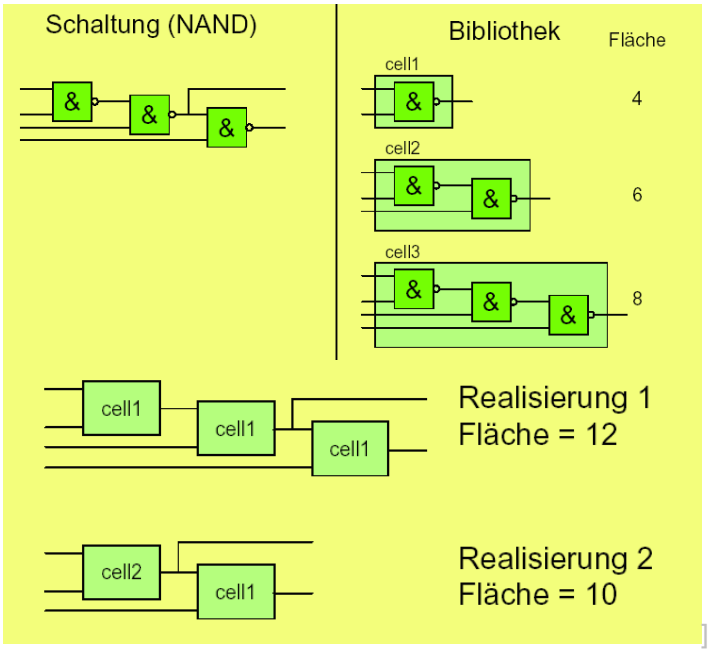
Operationswerkt mit MUXen *Operationswerk mit Bussen*



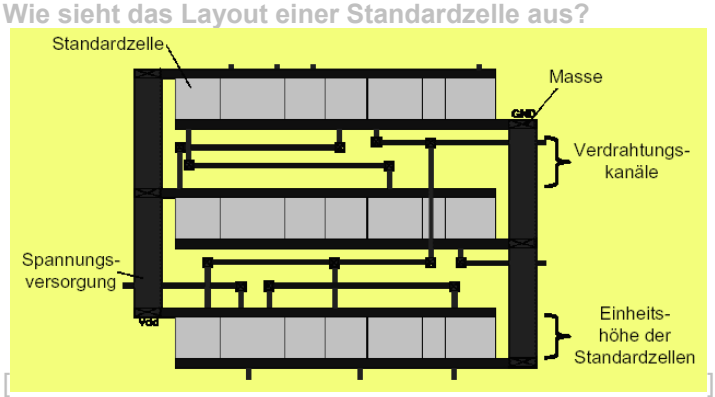
38. **Was geschieht bei der Synthese des Steuerwerkes?** [Der Ablaufplan des Steuerwerks wird in einen endlichen Automaten umgewandelt:



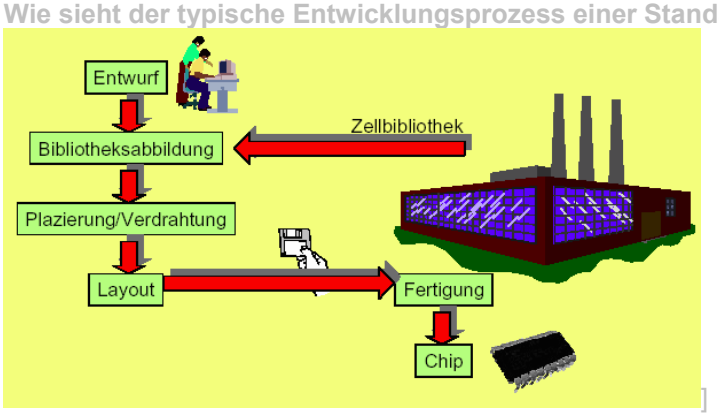
39. **Welche Realisierungsformen für Gatter gibt es?** [Standardzellen | Makrozellen➔ Standardisierte Funktionale Einheiten für Arithmetische Funktionen oder z.B. I/O Interfaces; PLAs; RAM/ROM | FPGA und natürlich diverse Mischformen]
40. **Was geschieht letztendlich bei der Gattersynthese?** [Die einzelnen RT-Module werden durch Gatterbeschreibungen ersetzt ➔ FSM-Synthese und Synthese aithmetischer Einheiten | Logikoptimierung, d.h. Minimierung und Retiming | Einfügen von Scanpfaden]
41. **Welche Möglichkeiten gibt es einen Logischen Ausdruck in Hardware umzusetzen?** [PLA/Bibliothekselemente/FPGA]
42. **Die Hersteller können ja nicht beliebige Bauteile als Fertiggatter bereitstellen, stattdessen gibt es fertige Biblitheken aus denen man sich beliebige Gatter selber „basteln“ kann. Wie geht das?** [Gegeben ist z.B. folgende Schaltung die mit Hilfe von Bibliothkeselementen zu realisieren ist:



43. **Wie sieht das Layout einer Standardzelle aus?**

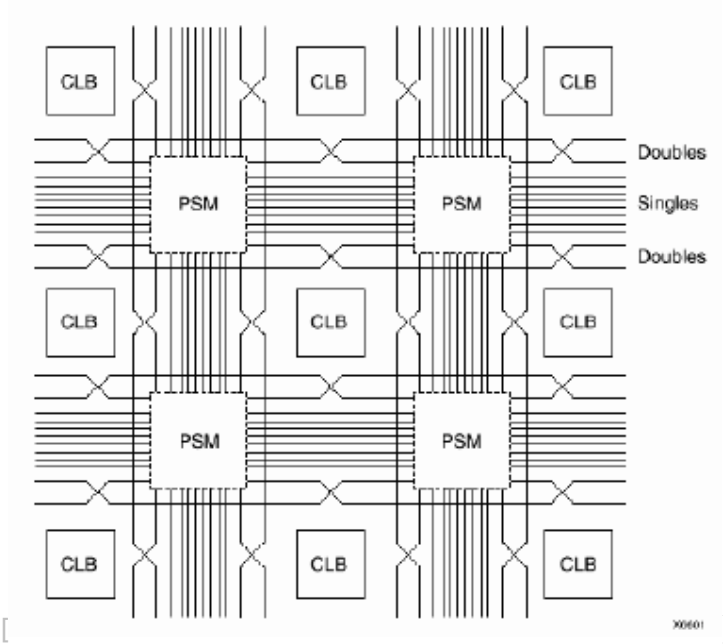


44. **Wie sieht der typische Entwicklungsprozess einer Standardzelle aus?** [



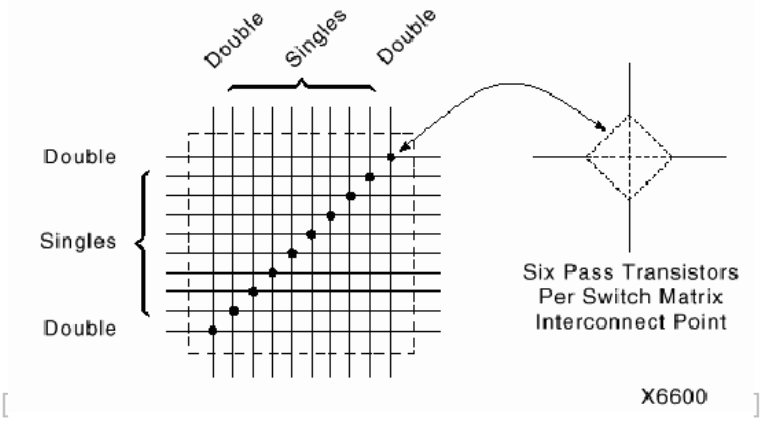
45. **Wie wird ein FPGA (Field programmable Field Array) realisiert und programmiert?**
[Alle Verbindungen und Logikblöcke sind programmierbar.
Beispiel am Xilinx 4000E:
Besteht aus einem Configurable Logic Block (CLB) und eine Programmable Switch Matrix (PSM).
➔ Der CLB besitzt zwei 4-Inpit-LUT (LookUpTables) und einem 3-input LUT, des weiteren aus zwei SR D-FlipFlops. Bypass-Pfade und eine Carry/Cascade Logic,
➔ Der PSM hat 10 Verbindungspunkte pro Matrix mit jeweils 6 Passtransistoren. Jede Verbindung ist zwischen 4 Richtungen möglich]

a. **Wie sieht die Architektur des Xilinx 4000E aus?**



[ ]

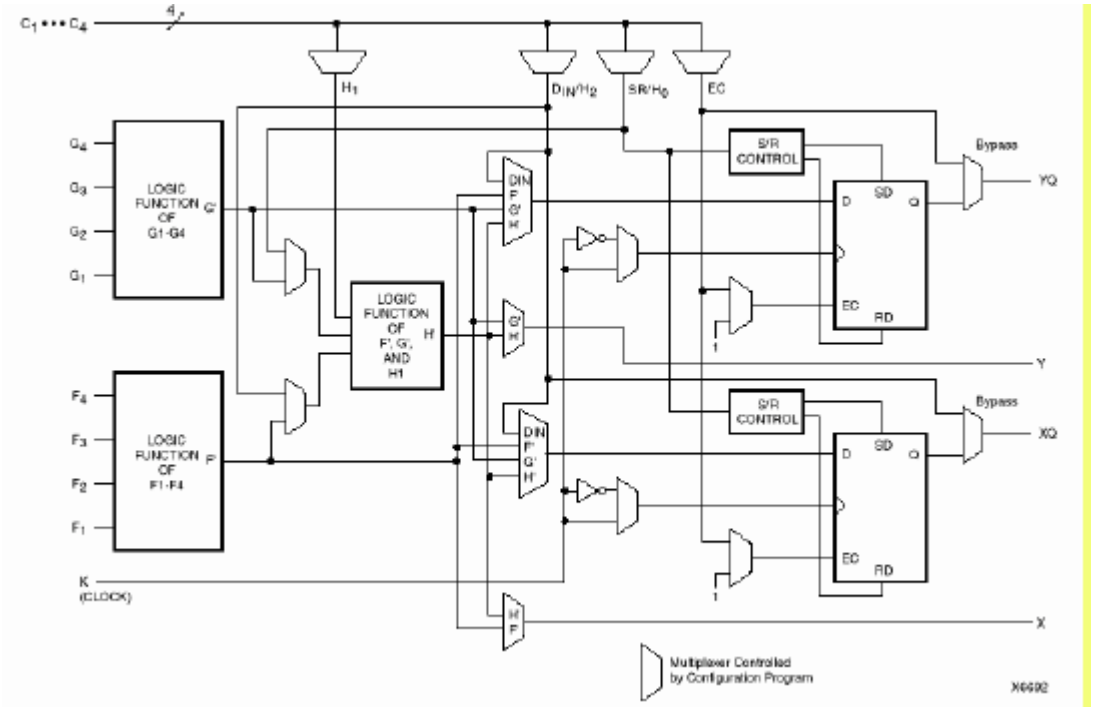
b. **Wie kann man sich das mit der PSM Verbindungsmatrix vorstellen?**



[ ]

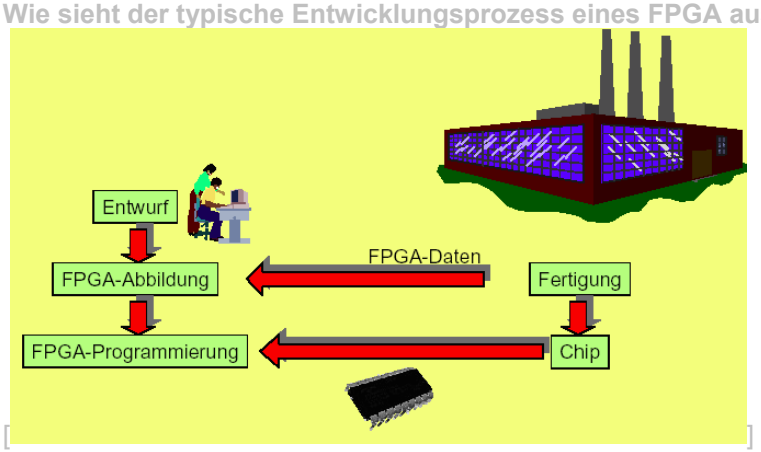
c. **Und wie sieht nun noch der CLB dazu aus?**

[



]

46. **Wie sieht der typische Entwicklungsprozess eines FPGA aus?**



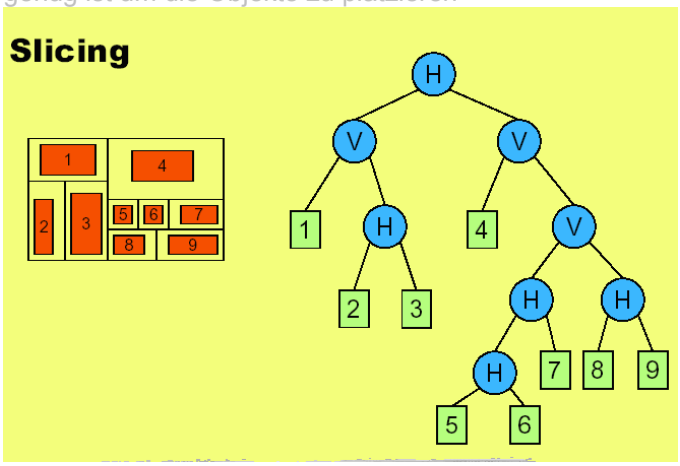
[ ]

47. **Was versteht man unter Partitionierung?** [Umfangreiche Schaltungen sind zu komplex um als Gatter platziert werden zu können, wir teilen deshalb das Gesamtproblem auf in einzelne Objekte. Das allgemeine Partitionierungsproblem ist leider NP-vollständig]
48. **Welche allgemeinen Partitionierungsverfahren kennen Sie?** [Da gibt es exakte Verfahren, wie „Enumeration der Lösungen“ oder „Integer Linear Programs (ILP)“, und es gibt heuristische Verfahren die konstruktiven Varianten „Random Mapping“ ider Hierarchical Clustering“ oder die iterativen Verfahren wie „Kernighan-Lin“, „Simulated Anealing“ oder auch evolutionäre Algorithmen]
49. **Wie erzeugt man gute Bipartitionen nach Kernighan-Lin?** [Man erzeugt zwei Bipartitionen der einzelnen Objekte. Nun werden diejenigen Objekte vertauscht die den größten Kostengewinn verursachen ➔ Einmal vertauschte Objekte werden nicht mehr vertauscht!]
    a. **Kann Kernighan Lin in lokalen Minima hängen bleiben?** [Nein, denn es ist auch erlaubt einen Kostenzuwachs zu erzeugen, wenn es ein Paar gibt, für welches es keinen Verringerung der Kosten gibt. Die Erhöhung muss allerdings trotzdem minimal bleiben]
    b. **Wie sieht es mit der Zeitkomplexität aus?** [$O(n^3)$➔ Bei partitionierung in m Blöcke dauert es $O(m*n^3)$]
50. **Was ist das Prinzip von Simulated Anealing?** [Beim abkühlen nehmen z.B. Metalle oder Glas einen Zustand minimaler Energie an. Da versucht man auf Minimierungsprobleme zu übertragen. Die Energie entspricht den Kosten einer Lösung. Die Kosten werden verringert

mit simulierter Temperatur. Eine Kostenerhöhung wird auch akzeptiert, allerdings nur zu einer bestimmten Wahrscheinlichkeit]

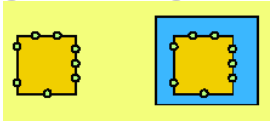
a. **Wann bricht das Verfahren ab?** [Wenn sich das so genannte Equilibrium einstellt, d.h. keine Verbesserung mehr erzielt wird.
b. **Wie ist die Zeitkomplexität bei Simulated Anealing?** [Verschieden! Grundsätzlich kann man sagen, dass das Ergebnis umso besser ist, je langsamer man den Abkühlungsprozess simuliert! Aber man gibt sich im Allgemeinen mit Lösungen die auf Verfahren mit Polynomieller Laufzeit beruhen zufrieden]

51. **Wie werden die jeweiligen Bipartitionen nun auf dem Träger platziert?** [Slicing muss eigentlich parallel zu Kernigan Lin ausgeführt werden. Man unteilt bei der Platzierung die Chipfläche iterativ in immer kleinere Bipartitionenen. In jedem Schritt werden die Objekte gemäße Kernighan Lin so vertauscht, dass der Verdrahtungsaufwand zwischen den Partitionen minimal ist. Hat man eine Bipartition komplett optimiert, wird jede Teilpartition wieder in zwei Bipartitionen unterteilt. Das macht man so lange, bis das Problem klein genug ist um die Objekte zu platzieren



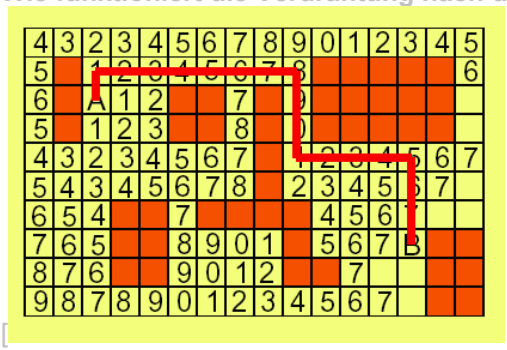
Abschließende werden in einem letzten Optimierungsschritt noch genau Positionen, Orientierungen und Seitenverhältnisse der Objekte festgelegt]

52. **Wieviel Platz muss man nun noch für die Verdrahtung der einzelnen Partitionen gemäß Slicing bereitstellen?** [Der sogenannte „halo“.:



ohne Halo mit Halo]

53. **Wie funktioniert die Verdrahtung nach dem Verfahren von Lee?**



[
Möchte man z.B. von einem Punkt A nach B verdraht, füllt man den gesamten umliegenden Rastern mit „Entfernungszahlen“ auf, ausgehend von A. Sobald man an B angelangt ist, sucht man sich von B aus rückwärts den besten Weg in dem man immer nach kleineren Zahlen schaut und diese bis zum nächsten „Anschlag“ verfolgt.]

54. **Kennen Sie weitere Verdrahtungsverfahren?** [Higtower/Channel-Routing(2Lag.)/River-Routing(1Lag.)]

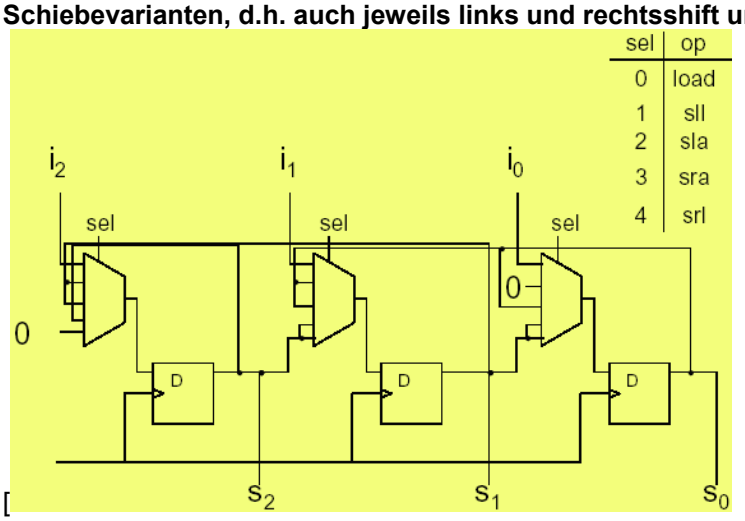
55. **Was ist der Unterschied zwischen logischem und arithmetischem Schieben?** [Der logische Shift überträgt die herausfallenden Bits in Carrybits und zieht NULLEN hinterher. Der arithmetische Shift entspricht einer Multiplitkation bzw Division mit 2. Es wird jeweils

das höchstwertige Bit von links nachgezogen.

- $srl_1(a_{n-1},\ldots,a_0) = (0,a_{n-1},\ldots,a_1)$
- $sll_1(a_{n-1},\ldots,a_0) = (a_{n-2},\ldots,a_0,0)$
- $sra_1(a_{n-1},\ldots,a_0) = (a_{n-1},a_{n-1},\ldots,a_1)$
- $sla_1(a_{n-1},\ldots,a_0) = (a_{n-1},a_{n-3},\ldots,a_1,0)$

]

56. **Wie ist die Schaltung eines Schieberegisters aufgebaut, welche alle 4 Schiebevarianten, d.h. auch jeweils links und rechtsshift unterstützt?**



[
Man erweiterte das ganz normale Schieberegister, welches einen logischen Rechtsshift ausführt um verschiedene Übertragsbits, die in Abhängigkeit der Eingabebits $i_0 \ldots i_2$ die gewünschte Art von Shift (gemäß der Tabelle) bewirken]

57. **Welche Zahlendarstellungen für ganze Zahlen kennen Sie?** [Die Positiven Zahlendarstellungen sind in allen gleich!. Die negativen unterscheiden sich:
Betrag-Vorz.: MSB="1"
Einerkomplement: MSB="1" und alle anderen Bits gekippt.
Zweierkompl.: wie Einerkomplement nur eine „1" dazuaddiert]

58. **Welche Vor- und Nachteile haben diese Darstellungen in Bezug auf Redundanz, Symmetrie und Berechenbarkeit?**

[

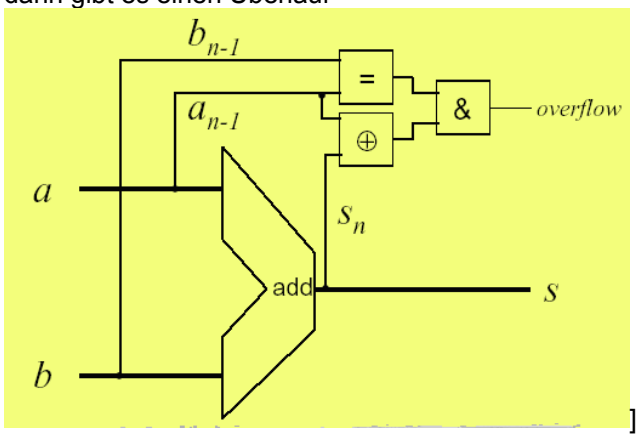
Darstellung	*redundant / irredundant*	*symmetrischer Zahlenbereich*	*geeignet zum Rechnen*
Betrag und Vorzeichen	**redundant**	**ja**	**bedingt**
Einer-Komplement	**redundant**	**ja**	**gut**
Zweier-komplement	**irredundant**	**nein**	**sehr gut**

]

59. **Welche verschiedenen Addierer kennen Sie?** [Conditional-Sum | Carry-Ripple | Carry-Chain | Carry Lookahead]

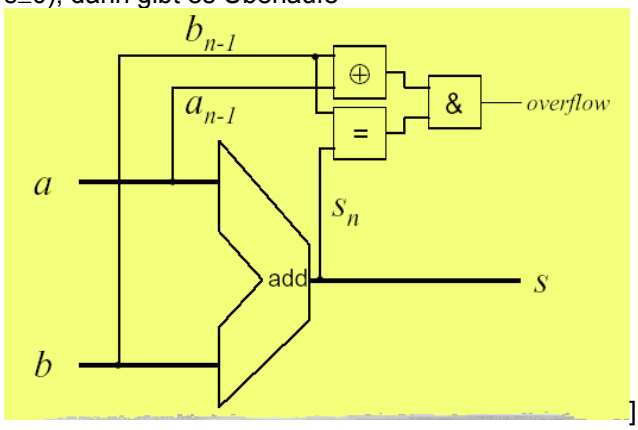
    a. **Wie kann man einen Überlauf beim Addieren erkennen? Welche Bedingung ist dann erfüllt?** [Wenn S=a+b, und a≥0 und b<0 bzw umgekehrt, dann gibt es keinen Überlauf! |
    Wenn a≥0 und b≥0 (➔Überlauf bei s<0) bzw. a<0 und b<0 (➔ Überlauf bei s≥0),

dann gibt es einen Überlauf



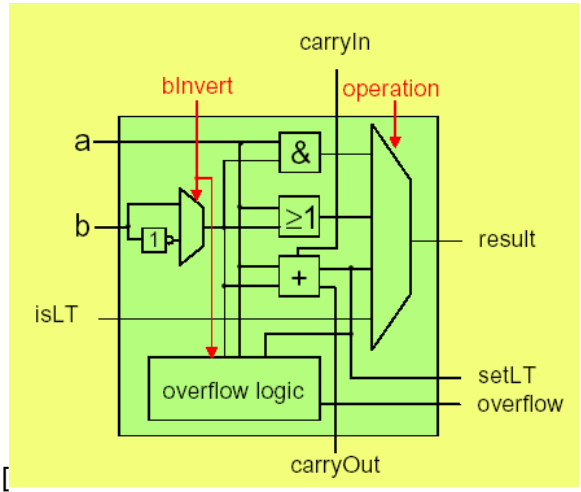
]

b. **Mit einem Addierer kann man üblicherweise auch subtrahieren, wie aber erkennt man hier einen Überlauf?** [Sei s=a-b. Wenn a≥0 und b≥0, dann gibt es keinen Überlauf!
Wenn a≥0 und b<0 (➔ Überlauf bei s<0), bzw. wenn a<0 und b≥0 (➔ Überlauf bei s≥0), dann gibt es Überläufe



]

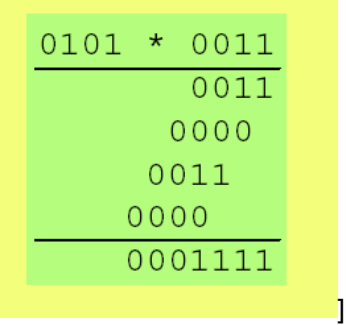
60. **Wie könnte eine einfache ALU aussehen?**



[
Wir möchten die Grundoperationen Addition, Subtraktion und Vergleich realisieren. Das einfachste Element ist die Addition. Für die Subtraktion müssen wir b invertieren können, für den Vergleich gilt: a<b gdw. a-b<0. Eine Overflow-Einheit sorgt für die Behandlung der Carrys.]

61. **Wie kann man zwei Zahlen im Zweierkomplement einfach nach der Schulmethode multiplizieren?** [Bei negativen Zahlen muss man zunächst den Betrag nehmen, bevor man multiplizieren kann. Leider ist das Verfahren ziemlich langsam, da n Additionen

durchgeführt werden müssen, insbesondere mit Null.



]

62. **Viel schneller kann man mit dem Verfahren von Booth multiplizieren. Welche Idee steckt dahinter?** [Nullen kann man Shiften: Wenn der Mulitplikator einen Nullblock der Länge k enthält, so kann man um k stellen shiften um die Multiplikation zu beschleunigen. Einserblöcke von Stelle u bis v kann man durch eine einzige Addition an Stelle v+1 und eine Subtraktion an Stelle u ersetzen. D.h. also, dass arithmetische Operationen immer nur an den 0→1 1→0 Übergängen stattfinden. Man erhält dadurch die Rechenvorschrift:

y_i	y_{i-1}	Operation
0	0	shift
0	1	add; shift
1	0	sub; shift
1	1	shift

mit $y_{-1}=0$

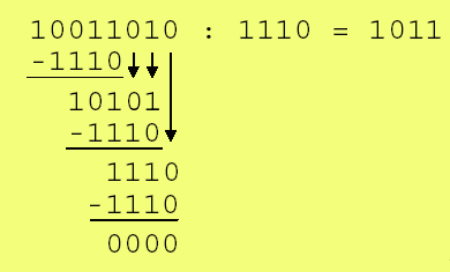
Folgendes Beispiel erläutert das ein wenig:

x	y_3	y_2	y_1	y_0	Operation	Zwischenergebnis
0010	0	1	1	0		0000
				↑	shift	0000 0
			↑		sub (add 1110)	1110 0
			↑		shift	1111 00
		↑			shift	1111 100
	↑				add 0010	0001 100
	↑				shift	0000 1100

Das Verfahren von Booth kann insbesondere auch negative Zahlen multiplizieren!]

a. **Nennen Sie eine worst-case Folge für das Verfahren von Booth!** [10101010101…]
b. **Muss man beim Verfahren positive bzw. negative Zahlen gesondert betrachten?** [Nein, das geht direkt!]
c. **Wie wird das Verfahren von Booth in Hardware realisiert?** [Einfach durch verwendung der Additionshardware der ALU. Der Nachteil ist allerdings die Geschwindigkeit, denn ein n-Bit Multiplikation benötigt mehr als n log n GAtterlaufzeitzen, was in anderen Hardwarerealisierungen besser ist (log n Gatterlaufzeiten)]

63. **Wie dividiert man zwei Zahlen nach der Schulmethode?**



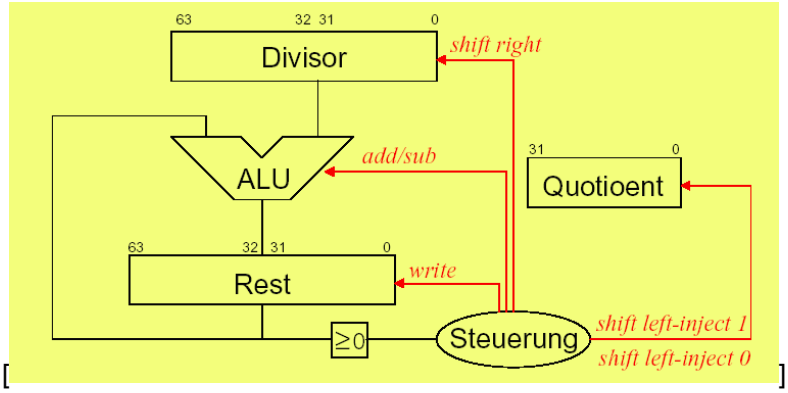
[ ]

64. **Wie unterscheidet sich die abgewandelte Schulmethode von der normalen?** [Im Prinzip gleich, nur dass man beim subtrahieren nicht gleich eine weitere Stelle hinzufügt, fall das Ergebnis kleiner als Null ist. Der Algorithmus führt die Subtraktion stur durch, und wenn er merkt, dass das Zwischenergebnis kleiner als Null war, wird eine Null im Gesamtergebnis eigetragen und die gemacht Subtraktion durch eine Addition rückgängig gemacht. Diesen Schritt nennt man Restore!

```
 10011010 : 1110 = 01...
-1110
 -101     <0
+1110     restore
 10011
 -1110
    101   ≥0
    ...
```

]

a. **Wie sieht das Rechenwerk für die Division aus?**

[

]

65. **Was versteht man bei der Carryoverflowbehandlung unter „Wrap around“ bzw. „Sättigungsarithmetik“?** [Wrap-Around: Überträge werde einfach weggelassen | Sättigungsarithmetik: Die Werte werden beibehalten

	Wrap-around Arithmetik	Sättigungs-Arithmetik
a	"1000"	"1000"
b	"1000"	"1000"
a+b	"0000"	"1111"

]

66. **Welche mathematischen Probleme existieren bei Festkommazahlen?** [keine großen/kleinen Zahlen darstellbar/Oprationen nicht abgeschlossen/Weder Assoziativ noch Distributivgesetz gelten]

67. **Wie werden Gleitkommazahlen üblicherweise dargestellt IEEE754?** [VZ|Exponent|Mantisse➔ Die Position des Kommas liegt nicht fest, deshalb der Name Gleitkomma bzw. Floating-Point

Gleitkommadarstellung einfacher Genauigkeit: $(-1)^S \cdot M \cdot 2^E$

31	30 29 28 27 26 25 24 23	22 21 . . . 3 2 1 0
S	Exponent E	Mantisse M

Gleitkommadarstellung doppelter Genauigkeit: $(-1)^S \cdot M \cdot 2^E$

63	62 61 . . . 53 52	51 50 . . . 3 2 1 0
S	Exponent E	Mantisse M

]

a. **Was bedeutet normalisierte Gleitkommazahl?** [Da die Darstellung von Gleitkommazahlen nicht eindeutig ist, gibt es eine normalisierte Darstellung: Definiton: Eine Gleitkommazahl ist normalisiert, wenn die Mantisse einen Wert zwischen 1 und 2 hat, also eine eins vor dem Komma. Diese 1 kann man als sogenanntes Hidden Bit weglassen, da es bei normalisierten Zahlen ja klar ist, dass es da ist. Der Mantissenwert ist also bei einer normalisierten Gleitkommazahl 1+ „Nachkommastellenbits“ aufsummiert.]
b. **Was ist der BIAS?** [Die Exponentenbits sind vorzeichenlos,d.h. um eine negative Zahl zu erhalten muss man den Zahlenbereich verschieben, dies geschieht mit dem BIAS, welcher bei einfacher Genauigkeit üblicherweise bei „127“ liegt und bei doppeltet Genauigkeit bei 1023. Der Exponent berechnet sich also so:

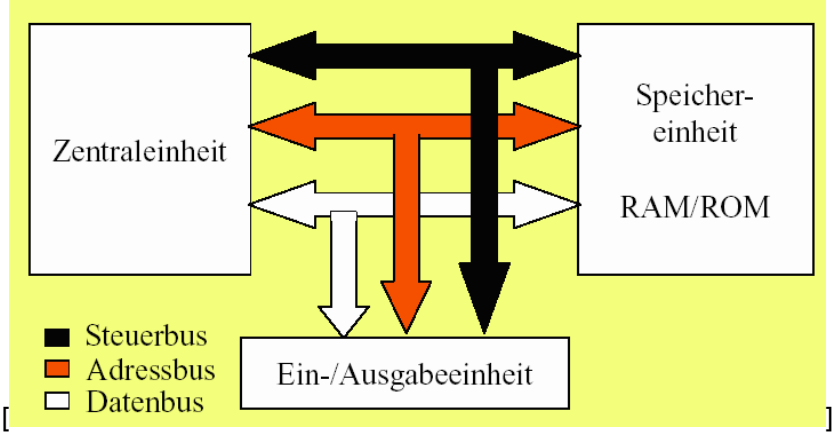
$$E = \sum_{j=0,\ldots,n-1} e_j 2^j - BIAS$$

]
c. **Was passiert, wenn der Exponent „0“ ist?** [Das Hidden-Bit wird als „0“ interpretiert! Man nennt diese Zahlen dann denormalisierte Zahlen, sie liegen zwischen 0 und der kleinsten normalisierten Zahl]
d. **Wie wird die Null dargestellt?** [Exp./Mantisse = 0]
e. **Wir wird der Wert unendlich dargestellt?** [Exp.=1/Mantisse=0]
f. **Welche Zahlen können denn nun mit IEEE754 dargestellt werden?**
[

	single precision	double precision
Vorzeichenstellen	1	1
Exponentenstellen	8	11
Mantissenstellen (ohne hidden Bit)	23	52
Bitstellen insgesamt	32	64
Bias	127	1023
Exponentenbereich	-126 bis 127	-1022 bis 1023
Darstellbare normalisierte Zahl mit kleinstem Absolutbetrag	2^{-126}	2^{-1022}
Darstellbare normalisierte Zahl mit größtem Absolutbetrag	$(1-2^{-24})\ 2^{128}$	$(1-2^{-53})\ 2^{1024}$
Darstellbare denormalisierte Zahl mit kleinstem Absolutbetrag	2^{-149}	2^{-1074}
Darstellbare denormalisierte Zahl mit größtem Absolutbetrag	$(1-2^{-23})\ 2^{-126}$	$(1-2^{-52})\ 2^{-1022}$

]
g. **Warum ist IEEE754 weder abgeschlossen noch gelten das Assoziativ- und das Distributivgesetz?** [Es sind ja nicht alle Zahlen darstellbar, und dann kann es eben passieren, dass man durch Anwendung der Gesetze den darstellbaren Zahlenbereich verlässt. Aus diesem Grund ist es eben auch bzgl. arithmetischer Operationen nicht abgeschlossen]
h. **Wie werden zwei IEEE754 Zahlen addiert?** [Die Exponenten müssen gleichnamig gemacht werden, d.h. der kleinere Exponent muss an den größeren angepasst werden. Dann werden die Mantissen addiert und ggf. wieder normalisiert]
i. **Wie werden zwei IEEE754 Zahlen multipliziert?** [VZ multiplizieren | Mantissen multiplizieren | Exponenten addieren und Bias subtrahieren | ggf. normalisieren]

68. **Wie ist ein „von Neumann“ Rechner aufgebaut?**



[ ]

   a. **Was ist das besondere hierbei?** [Gemeinsamer Programm und Datenspeicher]

69. **Was ist ein Bus, was ist der zentrale Vorteil?** [An einem Bus hängen alle Komponenten an einer Datenleitung, über welche gesteuert durch Interrupts Daten verschickt werden. Wenige Leitungen und sehr gut erweiterbar]
   a. **Welchen Nachteil gibt es?** [zu jedem Zeitpunkt kann nur ein Zugriff stattfinden]

70. **Welches sind die Aufgaben der CPU?** [Steuerung|Holen/Interpretieren/Ausführen von Befehlen|Organisation des Datenaustausches]
   a. **Aus welchen beiden Basikomponenten besteht die Zentraleinheit (CPU)?** [Steuerwerk für die Befehlsverarbeitung und ein Operationswerk für die Datenverarbeitung]

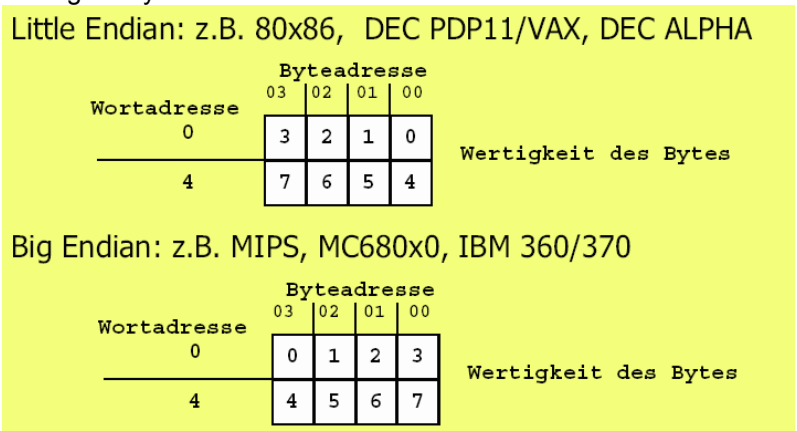
71. **Welche Datentypen kann der Prozessor verarbeiten?** [Bitvektoren|ganze Zahlen|Gleitkommazahlen]

72. **Was ist der Hauptspeicher eines Rechners?** [Der Hauptspeicher ist ein sehr großes addressierbares Array. In der Regel hat jedes Byte eine eigene Adresse, die direkt angesprochen werden kann. Es gibt aber Datentypen, wie z.B. Int, die von vorne herein aus mehreren Bytes bestehen. Schon aus Performance Gründen werdem einzelne Bytes gar nicht geschrieben, sondern immer größere Blöcke. Die Blöcke werden immer an Wortgrenzen ausgerichtet, d.h. aligniert.]
   a. **Welchen Vor- und Nachteil hat nicht aliginierter Speicher?** [Er ist zwar weniger speicherhungrig, es ist aber mehr Rechenaufwand erforderlich!

Objektadresse	Ausgerichtete Byteabstände	Nicht ausgerichtete Byteabstände
Byte	0,1,2,3,4,5,6,7	ausgeschlossen
Halbwort	0, 2, 4, 6	1, 3, 5, 7
Wort	0, 4	1,2,3, 5,6,7
Doppelwort	0	1,2,3,4,5,6,7

]

   b. **Welche beiden Arten gibt es die Bytes zu nummerieren (adressieren)?** [Ein Wort entspricht immer 4 Bytes: Little Endian ➔ der am wenigsten signifikante Teile enthält die niedrigste Byteadresse | big endian ➔ der signifikanteste Teil enthält die niedrigste Byteadresse



   Es gibt Rechner die auch beide Formate unterstützen. Für Strings sind Big Endians

natürlicher und für Zahlen little Endians, denn Strings werden von links nach rechts geschrieben, Zahlen hingegen umgekehrt, zumnindest was die Wertigkeit angeht]

   c. **Warum heissen die little- bzw. big endians so?** [little endian ➔ Weil die Byte-Adresse beim kleinsten Bit anfängt | big endian ➔ weil die Byte-Adresse beim größten Bit anfängt]

73. **Was versteht man unter Registerspeicher?** [kleiner,schneller als der Hauptspeicher. Wird benutzt um lokale Informationen in Registern zu halten und die Ausführung von Programmen zu beschleunigen.
   a. **Was versteht man unter homogenen Registern?** [Alle Register haben die gleiche Funktionalität➔ Adress- und Datenregister]
   b. **Was sind inhomogene Register?** [Spezialregister für die schnelle Ausführung spezieller Operationen]
   c. **Welche Spezialregister gibt es?** [ACCU➔ Akkumulator zum speichern eines Operanden, unterstützt die Ausführung von Schieben | PC➔ Programmzähler erzeugt durch inkrementieren die aufeinanderfolgenden Programmadressen bzw. lädt bei einem Sprungbefehl einen neuen Wert | Instruktions Register ➔ nimmt den aus dem Speicher gelesenen Befehl auf | Statusregister ➔ Informatione, die aus Operationen resultieren wie z.B. Überlauf, Übertrag oder Vorzeichen | Stackregister ➔ Aktuller Stand des Stackpointers]

74. **Nennen Sie einige verschiedene Befehlsklassen!** [Datentransportbefehle, arithmetische Befehle, logische Befehle, bitverarbeitende Befehle, Schiebe- bzw. Rotationsbefehlem, Stringbefehle, Sprungbefehle und |Systembefehle]

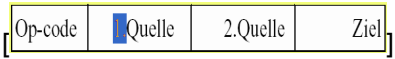
75. **Nennen Sie zu jeder Befehlsklasse ein Beispiel!** [
Transportbefehle: Transport oder besser gesagt Kopie eines Datums von Quelle zum Ziel
➔ z.B. MOVE B R0,R1 → Move Byte von Register R0 nach R1
Arithmetische Befehle: Festkomma Arithmetik für Byte, Word und Long
➔ ADD, ADDC (add with carry), SUB, SUBC, MULU, DIVU, ABCD (Add BCD Zahl), SBCD, INC, DEC, NEG, CMP (compare), Gleitkommaoperationen können entweder Softwaremässig durch Zerlegung oder Hardwaremäßig mit Co-Prozessor erledigt werden
Logische Befehle: Logische Funktionen
➔ OR, AND, XOR, NOT …
Bitverarbeitende Befehle: Betroffene Operandenbits werden durch die in einer Maske auf Eins gesetzten Bitpositionen addressiert
➔ BTST (test masked bits), BSET (test an set masked bits), BCLR (test and clear)
Schiebe und Rotationsbefehle: Verschieben eines Operanden um n Bitstellen
➔ Man unterscheidet logisches- arithmetisches- Schieben und Rotieren
Stringbefehle: Operation auf Byte- Wordketten
➔ MOVES (movestring), CMPS (compare string)
Sprungbefehle: Bedingt und unbedingte Sprungbefehle, Unterprogrammaufrufe und Rücksprung zum rufenden Programmabschnitt, Unterbrechung
➔ JMP → unbedingter Sprung |
➔ BRA → bedingter Sprung, nur wenn Bedingung erfüllt wird gesprungen z.B. BGT, BGE, BEQ…
➔ JSR → Sprung in Unterprogramm
➔ RTS → Rücksprung aus dem Unterprogramm
Systembefehle: Priviligierte Befehle (im Systemmodus) und Trap-Befehle (Softwaremässig erzwungener Übergang von Normalmodus in den Systemmodus
➔ Priviligierte Befehle: MOVESR (move status), MOVNSP (move normal stack pointer), STOP, RESET
➔ Trap-Befehle: MOVCC (move condition code) → zugriff auf Bedingungsbits, NOP, TRAP, TRAPV (trap on Overflow)]

76. **Was versteht man unter 0-stufiger bzw. 1-stufiger bzw. 2-stufiger bzw. mehrstufiger Speicheradressierung?** [
➔ 0-Stufig, implzit: das Register wird schon durch den Befehl implizit adressiert indem sich der Operand befindet (z.B. LSRA heisst verschiebe den Inhalt des Akkus um eine Bitposition nach rechts
➔ 0-Stufig, explizit: Das Register wird direkt im Op-Code angegeben (z.B. DEC R0 heisst „Dekrementiere den Inhalt von R[0]
➔ 0-Stufig, immediate: Operand ist im Befehl als Konstante enthalten (z.B. LD D3,#$A3 heisst D[3]:=$A3)
➔ 1-Stufig, direkt: z.B. LD D3,$A374 heisst D[3]:=M[$A374]

➔ 1-Stufig, Register indirekt: z.B. LD D3, (A4) heisst D[3]:=M[A4]
➔ 1-Stufig, indiziert: Effektive Adresse ergibt sich durch Addition eines „Index“ zu einem Basiswert
➔ 1-Stufig: Speicher-relativ: Basis absolute Adresse, Index im Register (z.B. ST R1,$A704(R0) heisst M[$A704+R[0]]:=R[1]
➔ 1-Stufig: Register-relativ: Basus in Basisregister, Index absolut (z.B. CLR $A7(B0) heisst M[B[0]+$A7]:=0
➔ Weitere Varianten nicht erläutert !!!!
➔2-Stufig: Ergebnis der ersten Berechnung liefert Adresse im Speicher, die wiederum Offset für folgende Adressberechnung enthält.
➔ Höherstufig gibt es nur in Ausnahmefällen!

Adressierungsart	Beispiel	Wirkung	Anwendung
Register	Add R4,R3	R4=R4+R3	Wert im Register
Immediate	Add R4,#3	R4=R4+3	Operand Konstante
Displacement/ Basisadress.	Add R4,100(R1)	R4=R4+M[100+R1]	lokale Variable
Reg.indirekt	Add R4,(R1)	R4=R4+M[R1]	Pointer (Zeiger)
Indiziert	Add R3,(R1+R2)	R3=R3+M[R1+R2]	Feld-Adressierung R1-Basis, R2-Index
Direkt/Absolut	Add R1,(1001)	R1=R1+M[1001]	statische Daten
Speicher-indirekt	Add R1,@(R3)	R1=R1+M[M[R3]]	Speicherplatz als Pointer (Wert=*p)
Autoinkrement	Add R1,(R2)+	R1=R1+M[R2] R2=R2+d	Für Zugriff auf Felder in Schleifen
Autodekrement	Add R1,-(R2)	R2=R2-d R1=R1+M[R2]	(wie Autoinkr.)
Skaliert/ Indiziert	Add R1, 100(R2)[R3]	R1=R1 +M[100+R2+R3*d]	Felder mit Daten der Länge d

77. **Warum gibt es so endlos viele Varianten der Adressierung?** [Man hat sich früher daraus einen kompakten Code erwünscht➔ Seit RISC gibt es nur noch wenige Arten, das Ziel des kompakten Programmcodes hat sich in das Ziel schneller Verarbeitung gewandelt! Bei SOC (System on Chip) System ist das aber immer noch wichtig, da sehr kleiner Speicher]
78. **Was versteht man unter einer 3-Adressmaschine?** [Man redet in diesem Fall auch von dyadischen Operationen, d.h. die Verknüpfung von zwei Operanden. Der Befehl ist ein 4-Tupel, bestehend aus der Art der Operation, der ersten Quelladresse für den ersten Operanden, der zweiten Quelladresse für den zweiten Operanden und einer Adresse in der das Resultat der dyadischen Operation gespeichert wird]



    a. **Wie sehen die charakteristischen Befehle aus?** [ ]
    b. **Was sind die Nachteile einer 3-Adressmaschine?** [Befehlsformat ist unhandlich, da sehr lang ➔ Ziel: Verringerung der Anzahl der Adressen innerhalb eines Befehls und damit Reduzierung der Länge des Befehls]
        i. **Wie kann man dieses Ziel erreichen?** [Verdeckte Adressierung, d.h. eine Quelle ist Quelle und Ziel zugleich | Implizite Adressierung: Registeradresse ist implizit im Operationscode enthalten, d.h. ein bestimmtes Register für eine Operation | Kurzadressen mit Basisregister und Displacement]
79. **Was versteht man nun unter eine 2-Adressmaschine, was ist der Vorteil gegenüber der 3-Adressmaschine?** [Im Gegensatz zur 3-Adressmaschine wird hier als Ziel die Adresse einer Quelle genommen, d.h. man spart eine Adresse und der Befehl ist kürzer]
80. **Wie kommt man nun zu einer 1 ½ Adressmaschine? Was wird dann noch im Parameter übergeben?** [Meist ist die erste Adresse ohnehin ein Register, dieses kann meist noch mit dem Befehlswort übergeben werden und muss nicht extra als Adresse angegeben werden, z.B.:]

opcode + regnr	op2
add R1,	$4A67
R1 := R1 + M[$4A67]	

81. **Es gibt auch eine 1-Adressmaschine, welche Aufageb hat hierbei der Akkumulator?** [Bei 1-Adressmaschinen ist die erste Adresse implizit im Befehl enthalten. Die übergebene zweite Adresse ist Quelle und Ziel zugleich. Der Akkumulator ist ein spezielles Register, dessen Inhalt nach Befehlsausführung überschrieben wird mit dem Ergebnis der Verknüpfung. Möchte man einen dyadischen Befehl ausführen, muss man das mit 3 1-Adressbefehlen realisieren]
82. **Was versteht man unter einer 0-Adressmaschine bzw Stackmaschine?** [ale arithmetischen Befehle werden einfach auf einen Keller (Stack) gelegt ohne diese explizit mit einem Operanden zu verknüpfen. Es werden immer die beiden obersten Elemente miteinander verknüpft!]
    a. **Beschreiben Sie, wie man damit z.B. addieren kann!** [2x Push mit den zu addierenden Adresse, dann ein ADD]
83. **Welche 4 verschiedenen Befehlstypen gibt es?** [
Register-Register Befehl ➔ Beide Operanden und das Resultat prozessorintern, der Befehl besteht nur aus einem Wort |
Register-Speicher Befehl ➔ Erster Operand aus Register, zweiter aus dem Speicher, das Ergebnis liegt im Speicher, der Befehl besteht aus 2 Worten |
Speicher-Speicher Befehl ➔ Beide Operanden aus dem Speicher, Befehl besteht aus 3 Worten
Speicher-Register Befehl ➔ Erster Operand aus Speicher, zweiter aus Register, Das Ergebnis steht dann im Register, der Befehl besteht aus 2 Worten]
84. **Welche beiden großen Mikroprozessorklassen gibt es?** [CISC = Complex Instruction Set Computer | RISC = Reduced Instruction Set Computer]
    a. **Gibt es weitere Klassen?** [DSP|EPIC VLIW]
    b. **Durch welche Eigenschaften zeichnet sich CISC aus?** [mächte Befehlssätze, komplexe Maschinenbefehle | viele Adressierungsarten| mehrere Datentypen | mikrokodierter Befehlssatz | wenige Register | Unterschiedliche Ausführungszeiten der Befehle]
        i. **Nennen Sie einige Argumente für/gegen die CISC Architektur!** [
        Vorteile: Komplexe Befehle schließen die noch offene semantische Lücke zwischen Soft- und Hardware | Die Diskrepanz zwischen interner Verarbeitungsgeschwindigkeit und Speichergeschwindigkeit ist reduziert. | Komplexe Befehle reduzieren den Speicherplatzbedarf im Hauptspeicher | Komplexe Befehle vereinfachen Compiler | Mehr zuverlässigkeit, frei nach dem Motto Hardware ist sicher, Software enthält Fehler |
        Nachteile: Hardware enthält leider genauso viele Fehler wie Software und ist zudem noch schwerer zu ändern | zB kein Pipelining]
        ii. **Welches Hauptargument gibt es gegen die Verwendung komplexer Befehle?** [werden kaum benötigt. Nur in Spezialfällen. Mehr als 90% der ausgeführten Befehle sind einfach. Die 10% der komplexen Befehle erhöhen i.d.R. die Ausführungszeit der einfachen Befehle. Lohnt sich das?]
        iii. **Welche bekannten, klassichen CISC Prozessoren kennen Sie?** [Intel 8086, Motorola 68000]
    c. **Durch welche Eigenschaften zeichnet sich die RISC Architektur aus?** [wenige, kleine Befehle, viel Compilerarbeit | in einem Zyklus ausführbare Befehle | viele Register | Leistungsfähige Speicherhierarchie | Wenige Adressierungsarten nötig (Load/Store). Verzicht auf Mikrocode da festverdrahtetes Steuerwerk | Schnelles Dekodieren der Befehle durch einfaches Format | Optimierende Compiler | Kürzere Entwurfszeit | Optimierung der Zeiteffizienz anstatt der speichereffizienz]
        i. **Nennen Sie Beispiele für Populäre RISC Prozessoren!** [MIPS RS2000/3000 Bereits 1982-89 ein 32 Bit Prozessor]
        ii. **Was bedeutet Load/Store Architektur?** [Es gibt nur zwei Befehle für den Speicherzugriff, nämlich Load [Speicherzelle] und Store [Speicherzelle]]
85. **Was sind Exceptions?** [Prozessorinterne Ereignisse, welche den normalen Programmablauf unterbrechen können]
    a. **Wann werden sie ausgelöst? Beispiele?** [z.B. arithmetischer Überlauf, Division durch Null, ungültiger Befehlscode]
    b. **Was passiert dann, wenn eine Exception ausgelöst wurde?** [Wenn z.B. der aktuell ausgeführte Prozessorbefehl die Ausnahmesituation verursacht, dann gibt es zwei Möglichkeiten: Entweder es wird in einem bestimmten Register ein Flag gesetzt,

welche vom Programm ausgewertet werden muss und entsprechend reagieren muss. Oder es wird eine definierte Ausnahmeroutine gestartet ➔ Interrupt]

86. **Was sind Interrupts?** [Ein externes Ereignis, welches den Programmablauf verändert, heisst Interrupt.]
    a. **Wann werden diese ausgelöst? Beispiele?** [Wenn asynchron zum Prozessor laufende externe Geräte kommunizieren wollen ➔ z.B. Eingabe über Tastatur/Maus, Grafikkarte, Datenübertragung Festplatte]
    b. **Was ist der Unterschied zwischen einer Exception und einem Interrupt?** [Eigentlich wurden die Interrupts nur für die Kommunikation mit externen Geräten eingeführt. Man kann diesen Mechanismus aber genauso gut verwenden um intern generierte Exceptions wie Hardwarefehlfunktionen zu behandeln | Exceptions sind im Übrigen synchron zum ausgeführten Programm und Interrups asynchron! Exceptions sind durch wiederausführung des Progammes reproduzierbar]
    c. **Was passiert, wenn ein Interrupt ausgelöst wurde? Beispiel?** [Nehmen wir das Beispiel, dass jemand eine Taste der Tastatur drückt, dann unterbricht die CPU das laufende Programm und startet den Interrupthandler des betreffenden Interrupts ➔ Lesen der Tastaturcodes, dann z.B. weitere Aktionen. Danach wird die Tätigkeit an der unterbrochenen Stelle fortgesetzt.]
    d. **Woher weiss das Betriebssystem welche Ursache ein Interrupt hat?** [Causregister ➔ Beim Auftreten eines Interrupts wird ein bestimmtes Bit (Flag-Bit) im Causregister gesetzt, welches vom Interrupthandler abgefragt wird | VectoredInterrupt ➔ Dem Betriebssystem wird ein Index aus der Interrupttabelle bereitgestellt, in der die Adressen für die speziellen Interrupts gehörenden Routinen gespeichert sind (ISR=Iterrupt Service Routine). Im Ausnahmefall wird die entsprechende Routine gestartet und durch den Befehl RTI (Return from Interrupt) verlassen. Exception Table z.B. von Intel:

INT	Function
0	Divide by zero
1	Single step (→ debugging)
2	Non-maskable interrupt (NNI)
3	Breakpoint
4	Overflow
5	Bound range exceeded
6	Invalid opcode
7	Coprocesor not available
8	Double fault exception
9	Coprocessor segment overrun
A	Invalid task state segment (?)
B	Segment is not present
C	Stack exception (Überlauf)
D	General protection exception
E	Page fault
F	Reserved
10	Coprocessor error

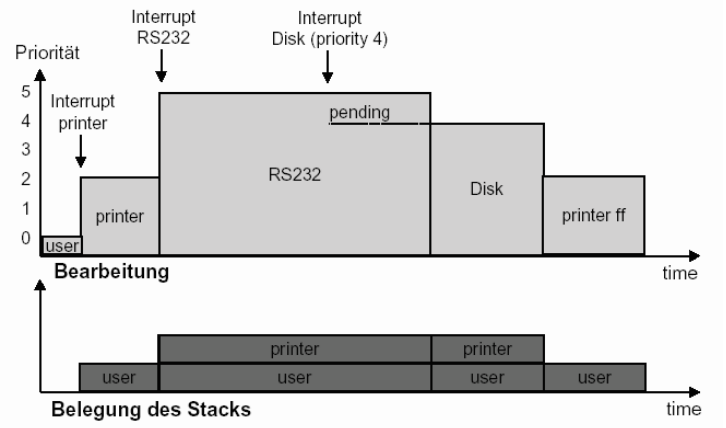
und Interrupt Tabelle:

INT	Function
0	Timer (55 ms – Intervall)
1	Keyboard
3	COM2 and COM4 (serieller Port)
4	COM1 and COM3 (serieller Port)
5	LPT2 (paralleler Port)
6	Floppy Disk
7	LPT1 (paralleler Port)
8	Real Time Clock (CMOS Clock)
C	PS2 – Mausport
D	Numeric Coprocessor Error
E	IDE0 – Festplatte / CDROM / ...
F	IDE1 – Festplatte / CDROM / ...

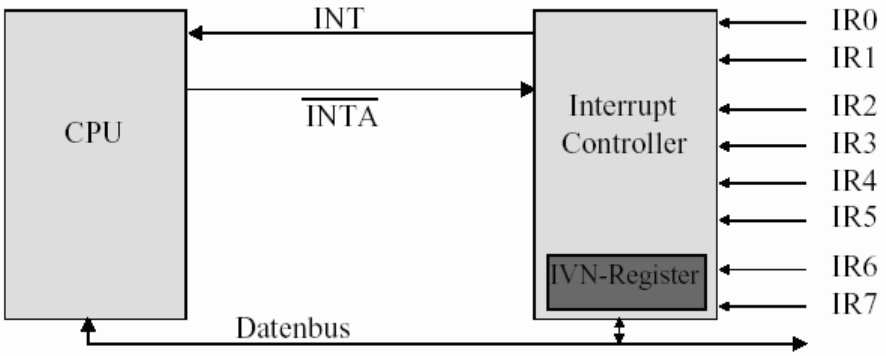
]

87. **Was versteht man unter Maskierten bzw. Nichtmaskierten Interrupts?** [Maskierte Interrupts werden nur dann ausgeführt, wenn das Interrupt Enable Bit in einem bestimmten Register gesetzt ist (d.h. der enstprechende Interrupt gesperrt ist) | Nichtmaskierte Interrupts werden hingegen immer ausgeführt, sie zeigen in der Regel Ausnahmesituationen an, die die Funktionalität des Systems gefährden]

88. **Was macht die Interrupt Service Routine, wenn während eines Interrupts ein weiteres Gerät einen Interrupt auslöst?** [ausmaskieren aller Interrupts | Prioritäten an die Interrupts vergeben ➔ Falls ein Device mit einer Priorität n einen Interrupt auslöst, so wird ein ISR mit dieser Priorität gestartet. Kommt ein Device mit einer höhern Priorität wird die aktuelle ISR unterbrochen und die neue sofort gestartet . Bei Devices mit niedriger Priorität wird der Interrupt in eine Warteschlange zur späteren Ausführung gestellt (Pending Interrupt



| Interruptcontroller ➔ Gibt es keine verschiedenen Interruptprioritäten, wie z.B. bei den Intel x86 Prozessoren, so benötigt man einen Interruptcontroller ➔ Dieser verwaltet dann die Prioritäten. Er erhält Rückmeldungen vom Prozessor, wenn Interrupts beendet sind



- I/O-Device *j* teilt dem Controller durch Setzen der Signalleitung IR*j* mit, daß eine Ausnahmesituation vorliegt.
- Die Kennung des Interrupts wird in das Register **IVN** eingetragen.
- Der Controller gibt die Meldung durch Setzen des Signals INT an die CPU weiter.
- Ist die CPU bereit, so teilt sie dies dem Controller durch Setzen des Signals INTA mit.
- Der Interrupt-Controller legt den Inhalt seines IVN-Registers auf den Datenbus.

- Der Interrupthandler der CPU nimmt den Inhalt des IVN-Registers vom Bus und rettet ihn in einem eigenen Register. Hiermit weiß der Prozessor, welches periphere Gerät den Interrupt ausgelöst hat.
- Vor der Abarbeitung des Interrupts rettet die CPU den Befehlszähler, sowie Steuer- und Statusregister auf dem Stack.
- Die CPU schlägt in der Interruptvektortabelle nach, um mit Hilfe der IVN die Startadresse der entsprechenden Interrupt-Serviceroutine (ISR) zu finden.

- Die CPU führt die ISR aus. Falls sie ISR Register verwendet, so muß sie deren Inhalt vorher auf dem Stack sichern.
- Vor Beendigung der ISR holt diese die geretteten Registerinhalte vom Stack.
- Die CPU stellt den ehemaligen Zustand (Befehlszähler, Steuer- und Statusregister) wieder her.
- Dem Interrupt-Controller wird die Abarbeitung des Interrupts mitgeteilt.

Interrupttabelle des PCs geordnet nach Prioritäten!

INT	function
0	Timer (55 ms Intervall)
1	Keyboard service
8	Real time clock
9	-- Einsteckplätze
10	-- Einsteckplätze
11	-- Einsteckplätze
12	PS2 – Mouseport
13	Numeric coprocessor error
14	IDE0 – Festplatte / CDROM / ...
15	IDE1 – Festplatte / CDROM / ...
3	COM2 oder COM4 (serieller Port)
4	COM1 oder COM3 (serieller Port)
5	LPT2 (paralleler Port)
6	Floppy Disk
7	LPT1 (paralleler Port)

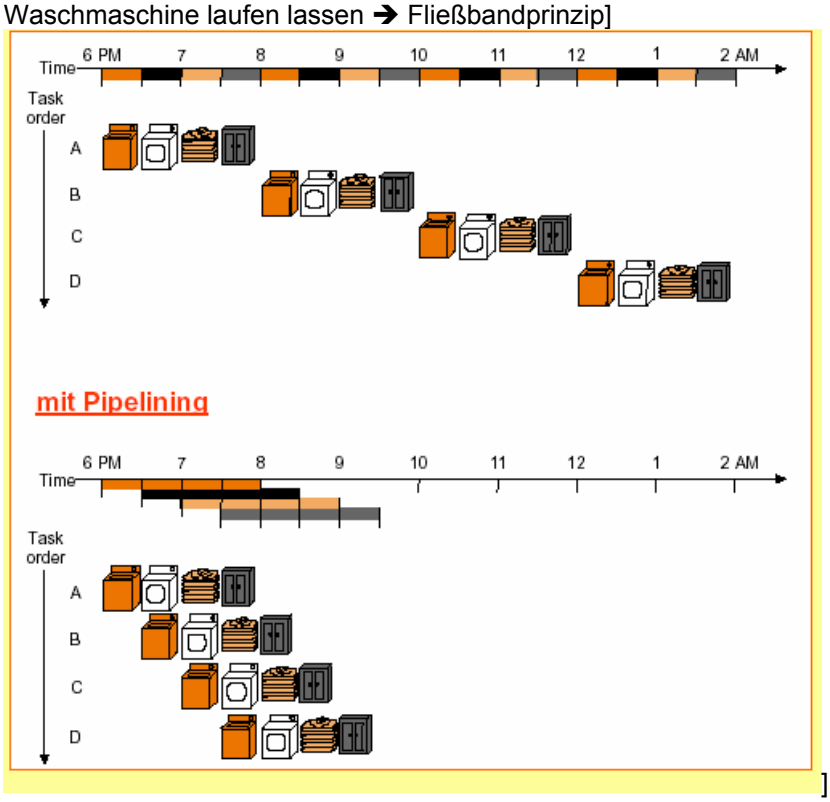
]

89. **Wie wird ein Maschinenbefehl wie z.B. „ADDD“ interpretiert?** [Durch ein Mikroprogramm, Es werden die einzelnen Phasen der Befehlsabarbeitung im Mikroprogramm Kodiert, d.h. Die Fetchphase (Holen des Befehls), Dekodieren des Befehls, Holen der Daten, Ausführen des Befehls, Schreiben des Ergebnisses]
    a. **Wie kommt man an das Mikroprogramm ran? Kann man da selber programmieren?** [Das Mikroprogramm ist im ROM vom Hersteller abgelegt werden. Man kann von aussen nicht darauf zugreifen. Jedem Prozessorbefehl ist ein bestimmtes Mikroprogramm zugeordnet]
    b. **Aus wie vielen Teilen besteht ein Mikroprogramm, wenn die Maschinensprache n Instruktionen umfasst?** [n+1: Zu jeder Maschineninstruktion ein Mikrocode und zusätzlich ein Programmblock zum Dekodieren der Befehle!]
90. **Wie wird die ALU angesteuert?** [Jede Operation (AND, OR, Substract, ADD…) die die ALU ausführen kann, hat einen bestimmten Code (ALU-Control)

ALUOp	Instruction opcode	Instruction operation	Funct field	desired ALU action	ALU control
00	lw	load word	xxxxxx	add	010
	sw	store word	xxxxxx	add	010
01	beq	branch equal	xxxxxx	substract	110
10	R-type	add	100000	add	010
	R-type	substract	100010	substract	110
	R-type	AND	100100	and	000
	R-type	OR	100101	or	001
	R-type	set on less than	101010	set less than	111

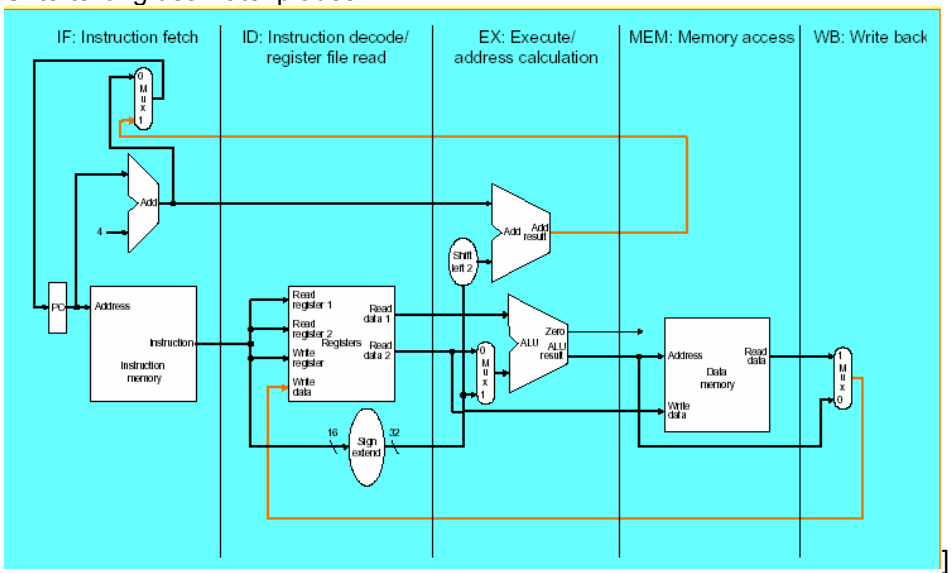
]
    a. **Was ist eine R-Type Instruktioen?** [Ein Register-Register Befehl]
91. **Die Alu wird nicht nur für artithmetische und logische Befehle eingesetzt! Bei welchen Befehlen wird sie noch verwendet?** [Load, Store ➔ durch add | beq ➔ Subtraktion zur Berechnung des Vergleiches]
92. **Erkläre das Pipelining Prinzip?** [Mit Pipelining kann man die Geschwindigkeit eines Rechners steigern, man nutzt dabei aus, dass der Prozessor bestimmte Teiloperationen sonst sequentliell ausführen müsste. Vergleichen kann man dies sehr schön mit Wäsche waschen. Wäsche waschen besteht z.B. aus 4 Schritten: Waschmaschine, Trockner, Bügelmaschine und Einräumen in Wäscheschrank! Wenn man eine Wäsche aus der Waschmaschiene holt und in den Trockner wirft, dann kann man bereits die nächste

Waschmaschine laufen lassen ➔ Fließbandprinzip]



]

a. **Wie könnten die Abarbeitungsschritte einer fünfstufigen Pipeline aussehen?** [IF | DECODE bzw. Lesen von Operanden aus den Registern | EXECUTE bzw. Adressberechnung | MEMORY ACCESS | WRITE BACK]
b. **Was sind die Vorraussetzung für Pipelining?** [Um Pipelining im Datenpfad ausnutzen zu können, muss die Abarbeitung eines Maschinenbefehls in mehrere Phasen gleicher Dauer aufgeteilt werden ➔ Abhängig vom Befehlssatz und der verwendeten Hardware . Schwer bei CISC, sehr gut bei RISC]
c. **Wie wird der Datenpfad z.B. in die 5 Phasen aufgeteilt?** [Logische und physische Unterteilung des Datenpfades:



]

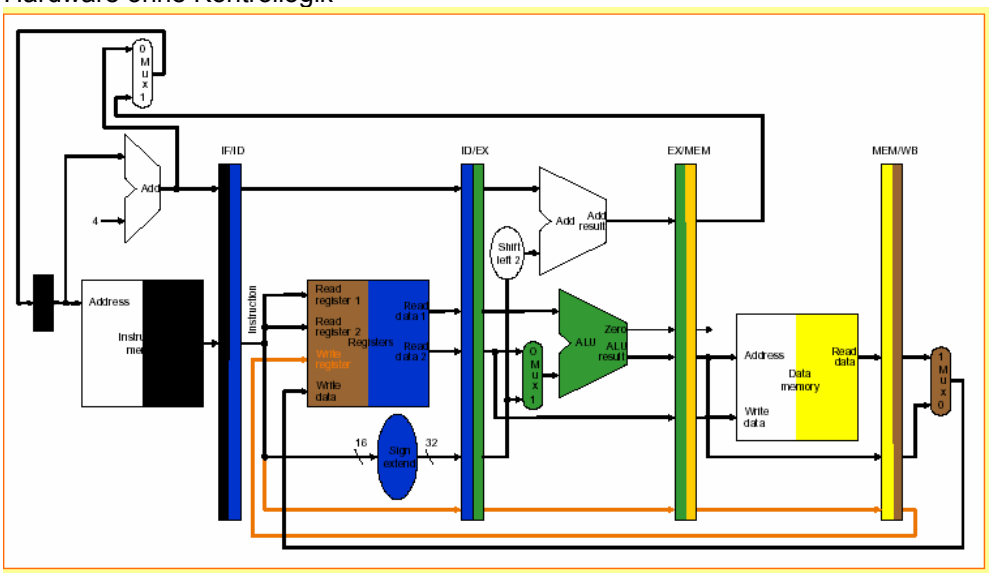
d. **Was ist der Pipelining Speedup?** [Die Beschleunigung ist das Verhältnis zwischen der Befehlsausführung mit Pipeline und ohne Pipeline.]
    i. **Wie berechnet sich dieser?** [Abarbeitungszeit eines Befehls ohne Pipeline ist t. Es gibt k Pipelinestufen, d.h. die Laufzeit einer Stufe der Pipeline ist t/k. Sei m die Anzahl der auszuführenden Instruktionen:
    m=1 ➔ Keine Beschleunigung, da Laufzeit =k*t/k=t ist
    m=2 ➔ Laufzeit ist t+t/k ➔ Beschleunigung ist 2k/k+1
    m≥1 ➔ Laufzeit ist t+ (m-1)t /k

Laufzeit ohne Pipeline ist mt
Daraus ergibt sich die Beschleunigung von

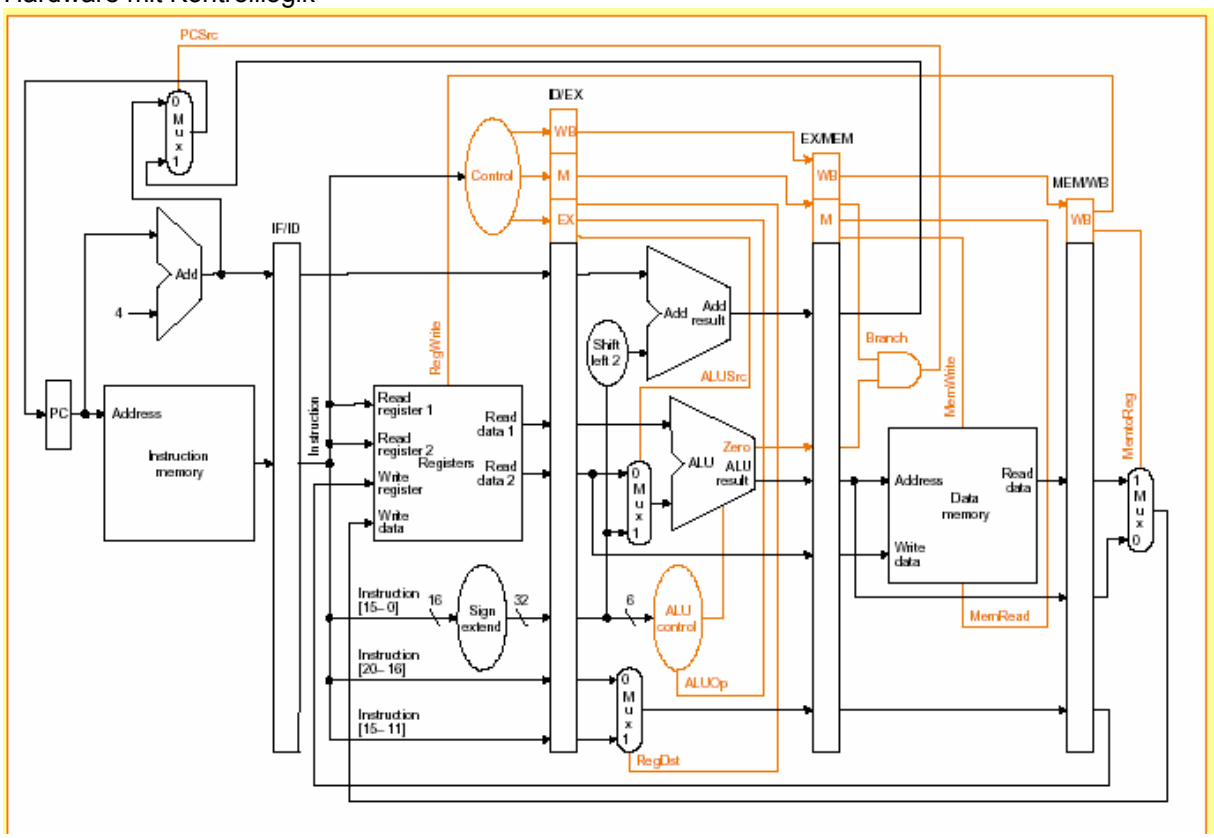
$$\frac{mt}{t+(m-1)\cdot\frac{t}{k}} = \frac{mk}{m+k-1} = k - \frac{k(k-1)}{m+k-1}$$

ii. **Welchem Wert nähert sich der Speedup an?** [k, also der Anzahl der Pipelinestufen. Unter der Vorraussetzung, dass sich die Befehle ohne weiteres verzahnen lassen]

93. **Für was benötigt man Registerbänke?** [Zwischenspeichern der Werte und Kontrollsignale einzelner Hardwarestufen der Pipeline um die Phasen hardwaremässg zu trennen
Hardware ohne Kontrollogik



Hardware mit Kontrolllogik



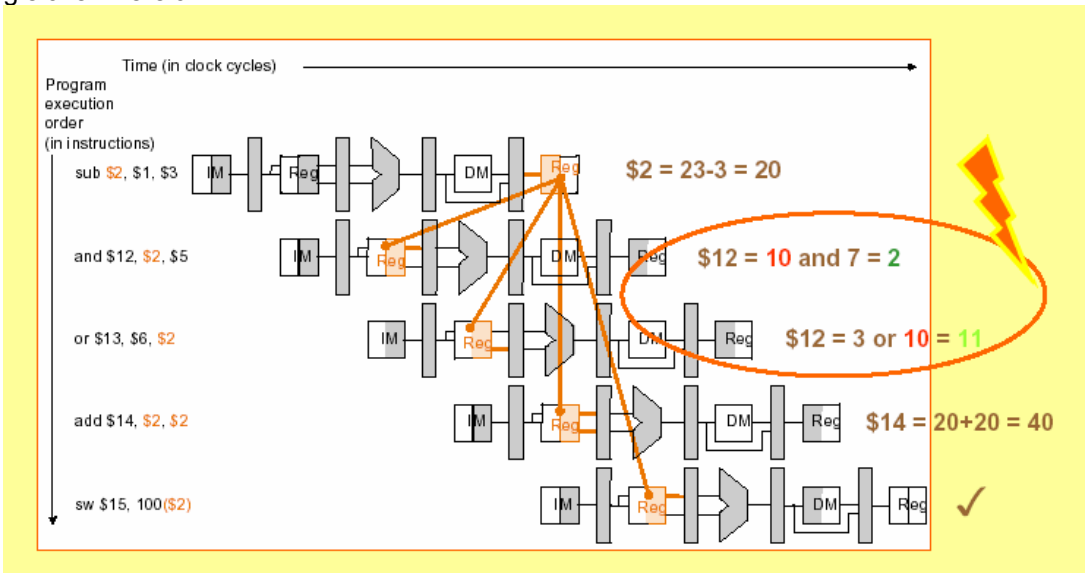
]

94. **Wann heissen zwei Anweisungen S_1 S_2 Datenunabhängig?** [Die DEF_i die Menge der Variablen auf die S_i schreibend zugreift und USE_i die Menge der Variablen auf S_i lesend zugreift, dann sind S_1, S_2 datenunabhängig, wenn

$$(DEF_1 \cap USE_2) \cup (DEF_2 \cap USE_1) \cup (DEF_1 \cap DEF_2) = \varnothing$$

]

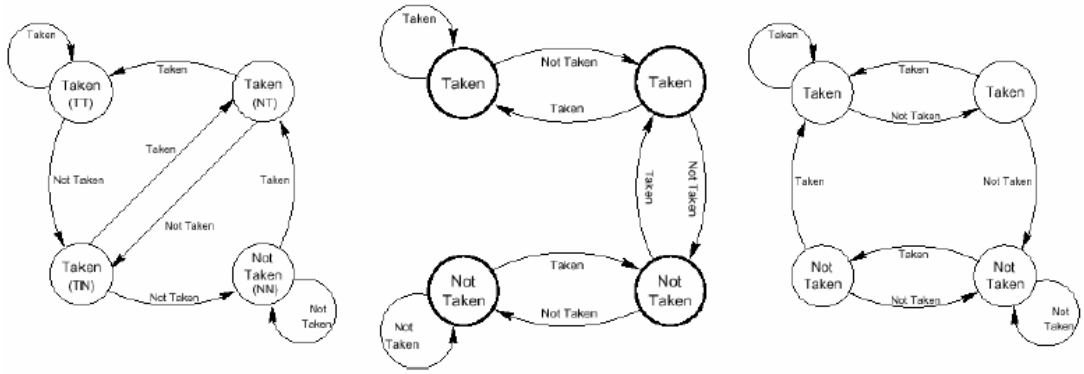
95. **Welche Probleme kann es in der Pipeline geben?** [Datenabhängigkeiten:
$DEF_1 \cap USE_2 \neq \varnothing$ ➔ True-Dependence ➔ Read after Write, d.h. S_1 schreibt etwas, was S_2 danach liest
$USE_1 \cap DEF_2 \neq \varnothing$ ➔ Anti-Dependence ➔ Write after Read, d.h. S_1 liest etwas, was S_2 kurz danach beschreibt
$DEF_1 \cap DEF_2 \neq \varnothing$ ➔ Output-Dependence ➔ Write after Write, d.h. beide schreiben in den gleichen Bereich



True-Dependence]

96. **Welche der drei Datenabhängigkeiten sind bei einer normalen Pipeline eher unproblematisch? Warum?** [Output- und Anti-Dependence, weil in der normalen Pipeline erst am Schluss geschrieben wird. Treten diese doch auf, z.B. bei Superskalarausfürhung oder bei einer Out-of-Order-Execution, dann kann diese der Compiler zum Teil durch Variablenumbennungen und Variablenkopien auflösen. Man nennt diese aufgelösten Dependencies dann Pseudo Dependence]
97. **Welche Aufgaben muss ein Compiler übernehmen?** [Entdecken von Hazards | Einfügen von NOPs]
98. **Was versteht man unter Result Forwarding?** [Ergebnisse aus der ALU gleich wieder an den Eingang der ALU legen und zusätzlich an den Speicher. Datum steht somit sofort zur Verfügung ➔ Evtl. ist die Ersparnis (der letzten Stufe) nicht gut genug und es steht evtl. doch noch der alte Wert im Register]
99. **Was ist Register Bypassing?** [Gelesener Wert aus dem Speicher wird direkt an die ALU geleitet, nicht nur über die Register]
100. **Was ist ein Kontrollhazards?** [Sprungbefehl vorhanden: Man nimmt zur Behandlung an, dass der Sprung (Branch) nicht ausgeführt und füllt die Pipeline entsprechend auf. Wird die Verzweigung doch ausgeführt, dann muss die Pipeline geleert werden]
101. **Wie kann man Kontrollhazards Softwaremässig bzw. Hardwaremässig vermeiden?** [Soft: Einfügen von NOPs | Delayed Branch |
Hard: einfrieren (stalling) | Spekulative Ausführung ➔ statische bzw. dynamische Sprungvorhersage]
102. **Wie funktioniert die Delayed Branch Technik?** [ein unproblematischer Befehl wird direkt nach dem Sprung ausgeführt und auf jeden Fall noch zum Ende gebracht. Der Sprung wird also bereits eine Anweisung früher erkannt. Die Umstellung macht der Compiler. Das Finden einer Anweisung ist recht einfach. Zwei Anweisungen geht gerade noch, aber 3 Anweisungen fast unmöglich!]

103. **Was ist statische bzw. dynamische Sprungvorhersage**? [Statische: ein spekulativer Sprung wird ausgeführt, wenn dies bereits im Opcode des Befehls steht | Dynamische: Vorhersagetabellen mit Erfahrungswerten werden gehalten (sehr gut, 97% Genauigkeit)]
104. **Wie geht das bei der dynamischen Sprungvorhersage genau?** [Per-Adress Vorgeschichte Vorhersahe an Hand von Vorhersagetabellen=Branch History Table | Globale Vorgeschichte ➔ Pfad durch die letzten m Blöcke wird als Information herangezogen. Die Ähnlichkeit zu anderen Sprüngen wird erkannt]
    a. **Wie werden die Vorhersageinformationen gespeichert?** [In einer Vorhersagetabelle, die entweder nur die letzte Sprunginformation (1-Bit) einer Sprungposition gespeichert hat oder die letzten n Sprünge (n-Bit)➔ Vorhersage über eine Heuristik, Speicherung im Schieberegister. Des Weiteren gibt es die Möglichkeit die Information in einem endlichen Automaten zu speichern (2-3-Bits➔ Je nach dem in welchem Zustand sich der Automat befindet entscheidet er was zu tun ist. Ein Update ergibt eine Zustandsübergangsfunktion die den Automaten im Zustand wechselt.]
    b. **Kennen Sie Beispiele für solche Enstcheidungsautomaten?**
    [



2-Bit History | 2-Bit sättigender Zähler | 2-Bit träger Automat

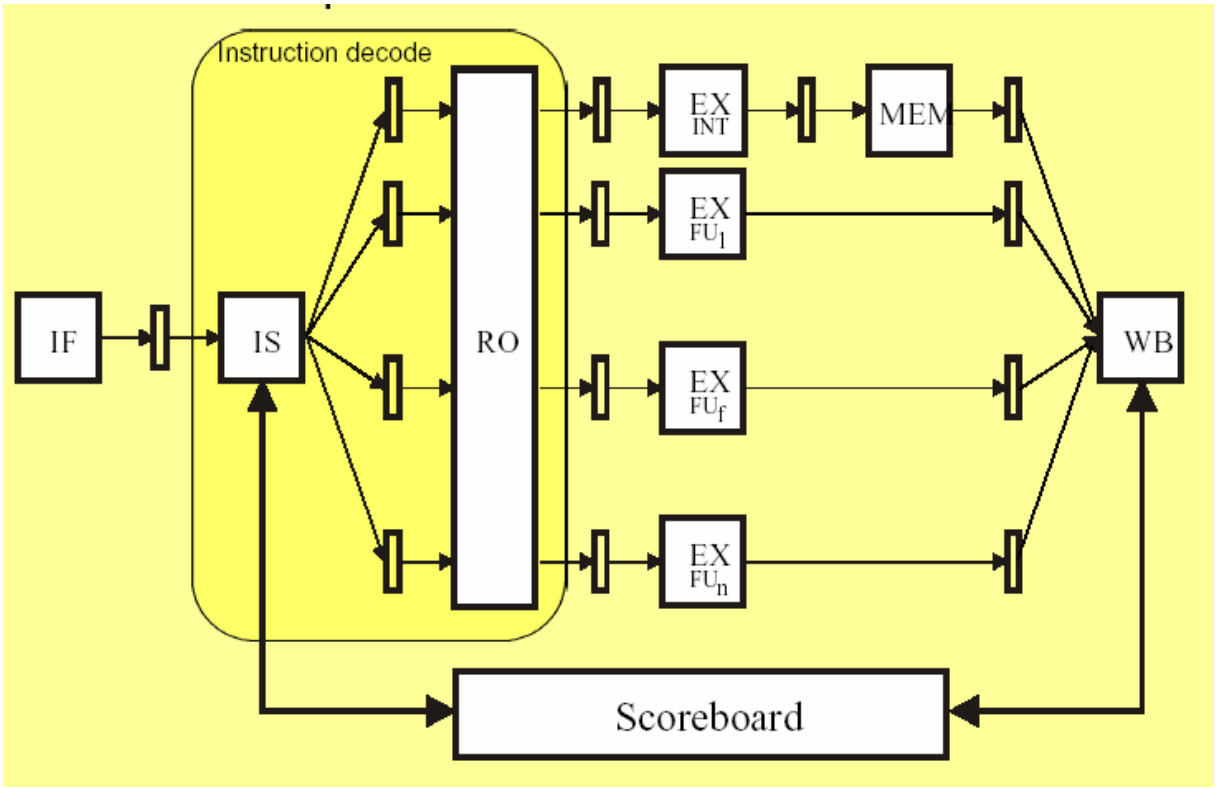
    Zähler und träge Automaten erzielen in der Regel die besten Vorhersagen bei geringstem Platz. Die Genauigkeit haengt auch vom Initialzustand ab]
    c. **Wie kann der Zugriff auf die Sprungzieladresse und auch die Befehle im Sprungziel beschleunigt werden?** [BTAC=Branch Target Address Cache➔Ein Cache welcher nach der ersten Ausführung eines Sprungbefehls die Adressen der Sprungziele speichert und beim nächsten mal sofort präsent hat. | BTC=Branch Target Cache➔ Speicherung der nächsten k Befehle im Sprungziel. D.h. man kann schon Informationen über das Sprungziel einholen bevor man gesprungen ist]
105. **In einer Erweiterung der Pipelining Technik, wird die Execute Phase nocheinmal detailierter zerlegt. Welchen Vorteil bringt dieses Vorgehen mit?** [Die Execute Phase wir in funktionale Einheiten zerlegt wie z.B. eine Integer Einheit, eine Einheit zum Multiplizieren von FP und Integers oder auch FP Adder. Diese Funktionalen Einheiten stellen nun auch Stufen der Pipeline dar, allerdings können diese parallel arbeiten. Die Executeeinheiten sind allerdings in der Regel unterschiedlich lang. Der Vorteil ist aber, dass wenn eine Operation gerade eine lange FP division ausführt, können andere Operationen gleichzeitig die anderen Einheiten nutzen und sind nicht blockiert]
    a. **Welche Probleme bringt das mit sich?** [Bezüglich Befehlszuordnung, Ausführung und Befehlsbeendigung kann passieren, dass diese zwar nacheinander beginnen, aber in anderer Reihenfolge beendet werden, je nach dem wie lange die jeweiligen Execute Phasen sind. Man unterscheidet also zwischen in-order und out-of order (issue,execution, completion. Nei superskalarer Ausfürhung kann es auch sein, das gleichzeitig mehrere Befehle in die Execute Phase gewiesen werden

MULTD	IF	ID	*M1*	M2	M3	M4	M5	M6	**M7**	MEM	WB
ADDD		IF	ID	*A1*	A2	A3	**A4**	MEM	WB		
LD			IF	ID	*EX*	**MEM**	WB				
SD				IF	ID	*EX*	*MEM*	WB			

    ]

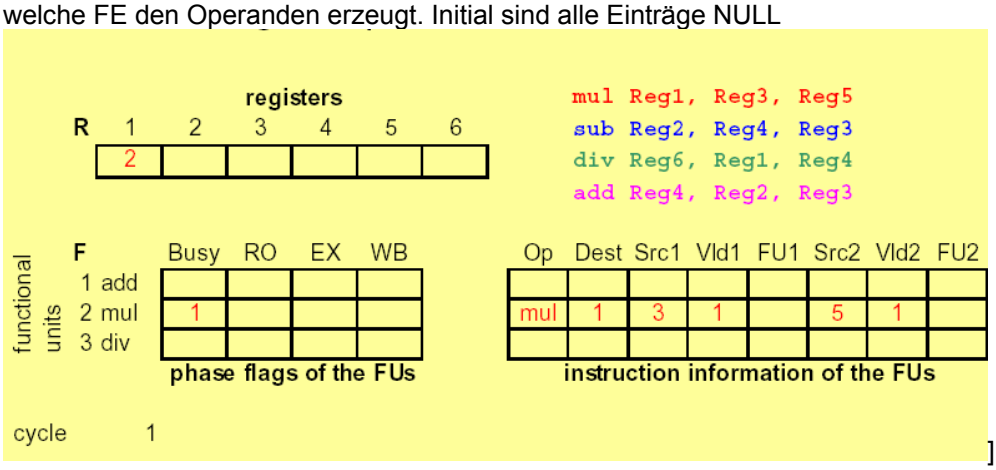
106. **Um dem nun entstehende Chaos Herr zu werden gibt es Scheduling. Was ist das?** [Scheduling ist ein Verfahren, das entscheidet, wann eine Instruktion gestartet wird, seine Operanden liest und das Ergebnis zurückschreibt. Das Ziel ist es Instruktionen so umzuordnen, dassdie Daten und Kontrollabhängigkeiten keine Rolle mehr spielen.]
    a. **Wie unterscheiden sich dynamisches und statisches Scheduling?** [Beim statischen Scheduling muss bereits der Compiler beim Übersetzen des Programmes eine geeignete Ausführungsreihenfolge berechnen | Dynamisches Scheduling wird direkt von der Hardware ausgeführt. Es wird immer versucht Instruktionen auszuführen die keine Datenabhängigkeiten zu andere Instruktionen haben ➔ Out of Order Execution und Completion wird erlaubt. Instruktionen werden direkt hinter „stall“ (Wartezyklen) ausgeführt.]
    b. **Dynamische Scheduling wird sowohl auf den Kontrollfluss als auch auf den Datenfluss angewandt. Wann und ggf. wie?** [Control Flow Scheduling wird zentral bei der Dekodierung vorgenommen (Scoreboarding) | Dataflow Schelduling wird in den einzelnen funktionalen Einheiten zur Laufzeit ausgeführt ➔ Befehle werden dekodiert und reservation stations zugeteilt, bis ihre Operanden verfügbar sind (Tomasulo Algorithmus)]

107. **Was ist nun genau Scoreboarding?** [Beim dynamic Scheduling: Globale Zustandsspeicherung der Pipeline ➔ welche Befehle sich in der Ausführung befinden, welche fuktionalen Einheiten aktuell benutzt werden und in welche Register die Ergebnisse gespeichert werden. | Befehle werden immer dann out-of order ausgeführt, wenn alle nötigen Ressourcen zur Verfügung stehen und keine Datenabhängigkeiten existieren.



]
    a. **In welche beiden Teile unterteilt sich die Dekodierphase**? [Zuordnungsphase (IS=Issue Phase)➔ Dekodieren und auf Struktur bzw. WAW-Hazards geprüft | Read Operands Phase (RO) ➔ Warten bis keine Datenhazards mehr existieren, dann erfolgt das Lesen der Daten aus den Registern]
        i. **Was sind Struktur Hazards?** [Wenn eine funktionale Einheit zu einem Zeitpunkt mehrfach benötigt wird]
    b. **Auch die Execute und die Writeback Phase müssen dann erweitert werden, welche Informationen müssen diese an das Scoreboard liefern?** [Execute Phase muss mitteilen, wann die Ergebnisse der fertig sind (result ready flag) | In der Writebackphase wird dann noch auf WAR Hazards geprüft und falls nötig die Registeraktualisierung gestoppt. Andernfalls wird die FE aufgefordert das Ergebnis im Register zu materialisieren]

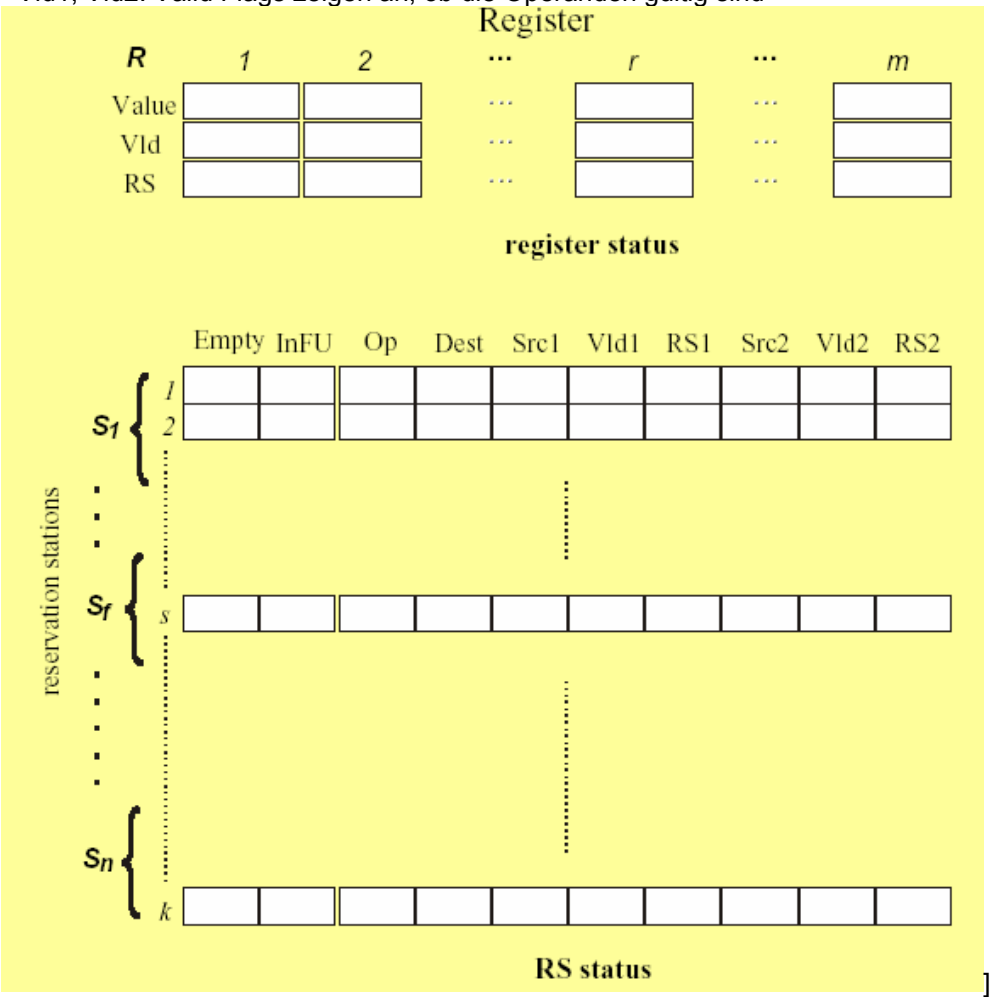
c. **Welche Tabellen muss Scoreboarding zur Ausführung unterhalten?** [
Register result status Tabelle (Regsiter) ➔ speichert, welche FE ein Ergebnis in welches Regsiter schreibt, d.h. die Anzahl der Einträge in R ist gleich der Anzahl der Register
Functional Unit status Tabelle (Functional) ➔ Speichert die Bearbeitungsphase jeder Instruktion, d.h. den aktuellen Zustand für jede FE, also Busy. RO, EX, WB
Instruction Status Tabelle (Functional) ➔ Ein Eintrag pro FE. Der Eintrag enthält: Den Opcode der bearbeiteten Instruktion, die Quell und Zielregister, die verfügbarkeit der Quellregister. Ist ein Operand gerade nicht verfügbar, wird zwischengespeichert, welche FE den Operanden erzeugt. Initial sind alle Einträge NULL



]

108. **Wann kann man zwei Register konfliktfrei umbenennen?** [Das Umbenennen der Register löst Namensabhängigkeiten auf, nämlich wenn z.B. zwei Instruktionen das gleiche Register benutzen, aber keine Daten zwischen den Instruktionen ausgetauscht werden. Durch diese Technik werden WAW und WAR Hazards vermieden.Das Umbenennen kann bereits statisch vom Compiler erfolgen oder auch dynamisch direkt von der ausführenden Hardware. Lösen kann man das Regsiterrenaming mit dem Tomasula Algorithmus.]
109. **Was macht der Tomasulo Algorithmus?** [Verteilte Zustandsspeicherung in den Reservation Stations | Common Data Bus (CDB) | Verhindert WAR und WAW Hazards | Register sind kein Flaschenhals | dynamisches Scheduling | Register Renaming | CDB ist der Flaschenhals]
    a. **Was sind die Reservation Stations?** [Jede FE hat eine oder mehrere solcher Reservation Stations. Die Reservation Station hält folgenden Inhalte gespeichert:
    - Bereits zugewiesene Instruktionen die auf Ausführung in der FE warten
    - Die Operanden des Befehls, soweit diese schon berechnet sind, ansonsten wird ersteinmal nur die Quelle (d.h. die FE die den Operaden gerade bearbeitet) des Operanden vermerkt
    ➔ Vermeidung von WAR Hazards, da die benötigten Register bereits in der Reservation Station liegen und das Überschreiben durch die andere Instruktion keine Rolle mehr spielt
    ➔ Vermeidung von WAW Hazards, da die Referenzen auf die Reservation stations zeigen und nicht auf die konkreten Register]
    b. **Welche Informationen werden in den Einträgen der Reservation Stations gespeichert?** [
    - Empty: Zeigt an ob die Reservation Station leer ist
    - InFU: zeigt an ob die FE die gespeicherte Instruktion gerade ausführt
    - Op: Operation die von der FE auszuführen ist
    - Dest: Tag des reservierten Registers
    - Src1,Src2: Werte der Quelloperanden
    - RS1, RS2: Tag der Reservation Station, die den Operanden erzeugt

- Vld1, Vld2: Valid Flags zeigen an, ob die Operanden gültig sind



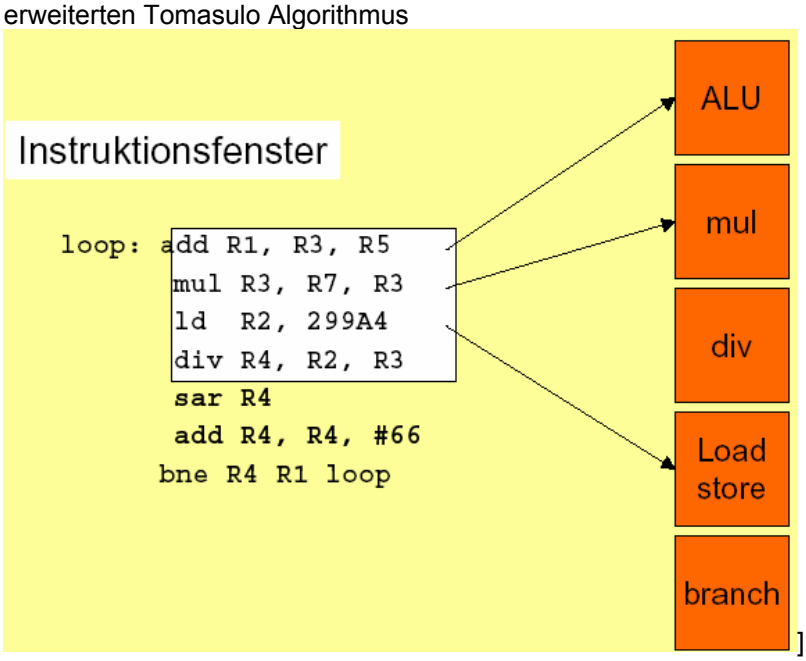
]

c. **Welche Aufgaben hat der Common Data Bus (CDB)?** [Nach der Ausführung eines Befehls schickt die RS eine Ergebnistoken über den Bus an die Register und alle anderen RS. Die RS überwachen ständig den CDB und lesen benötigt Ergebnisse vom Bus. Dies kann auch simultan von mehreren geschehen]

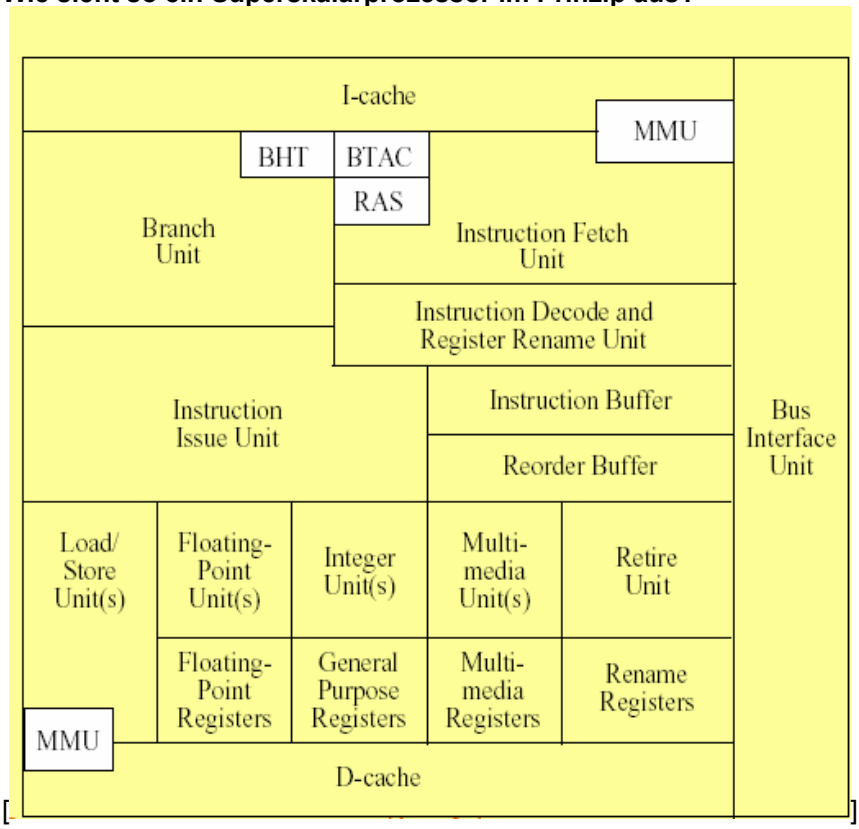
d. **Welche 3 Phasen gibt es beim Tomasulo Algorithmus?** [
1. Issue, d.h. Instruktion laden aus der Instruktion Queue
➔ Falls ein RS frei ist, wir die Instruktion zugewiesen und die Operanden geladen, falls möglich → In Order Issue
2. Execution, d.h. Ausführen des Befehls
➔ Wenn die Operanden beide zur Verfgung stehen, dann wird die Operation an die FE geleitet und dort ausgeführt. Stehen die Operanden noch nicht zur Verfügung wird der CDB abgehört bis die Daten dort erscheinen→Out-of-Order dispatch/execution
3. Write result, d.h. Beendigung der Ausführung
➔ Lege Ergebnis auf den CDB und gebe die RS frei]

110. **Was unterscheidet einen Superskalarprozessor von einem Single Instruction Prozessor?** [Mehrere Instruktionen können parallel ausgeführt werden. Die meisten heutigen Mikroprozessoren ordnen 4-8 Befehle gleichzeitig an die FEs zu mit dem

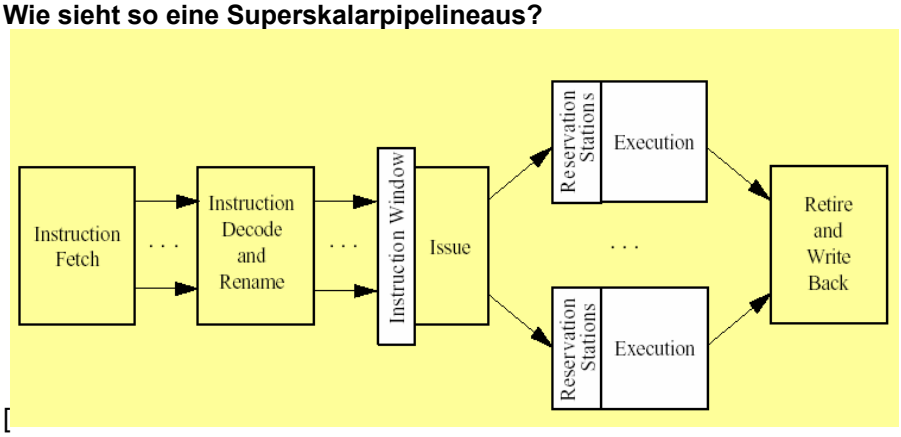
erweiterten Tomasulo Algorithmus



a. **Was versteht man unter Zuordnungsbandbreite?** [die maximale Anzahl von Befehlen die gleichzeitig zugewiesen werden kann]
b. **Wie sieht so ein Superskalarprozessor im Prinzip aus?**

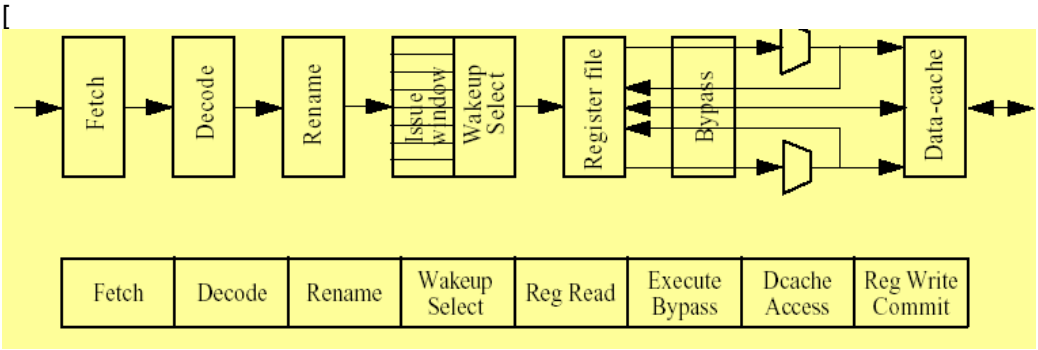


c. **Wie sieht so eine Superskalarpipelineaus?**



[
Wenn die Instruktionen im Instruktionsfenster, sind sie bereits frei von Kontroll und Namensabhängigkeiten (dies wird schon eine Stufe zuvor erledigt). Nur echte Datenabhängigkeiten (True-Dependencies) und Strukturkonflikte müssen behandelt werden]

d. **Wie kann eine Superskalarpipeline ohne Reservation Stations aussehen?**
[



]

e. **Was versteht man unter Issue (Zuweisung)?** [Die Zuweisungslogik untersucht die wartenden Befehle im Befehlsfenster und weist sie simultan den FEs zu-. Die Progammreihenfolge der zugfewiesenen Instruktionen wird im Reorder Buffer gespeichert (in-order vs. Out-of-order). Die Zuweisungslogik berücksichtig Datenabhängigkeiten und Ressourcenbeschränkung ➔ Dies kann in mehreren Stufen geschehen,d h. erst Prüfen der Strukturkonflikte und dann Prüfen der Datenabhängigkeiten (evtl. erst in Reservation Station)]

f. **Was versteht man unter dem Dispatch einer Instruktion?** [Die Übermittlung der Instruktion von einer RS an die FE, sobald alle Operanden der Instruktion zur Verfügung stehen. Dispatch ist keine Pipelinestufe. Dispatch und Ausführung werden nicht zwingend in der Programmreihenfolge durchgeführt]

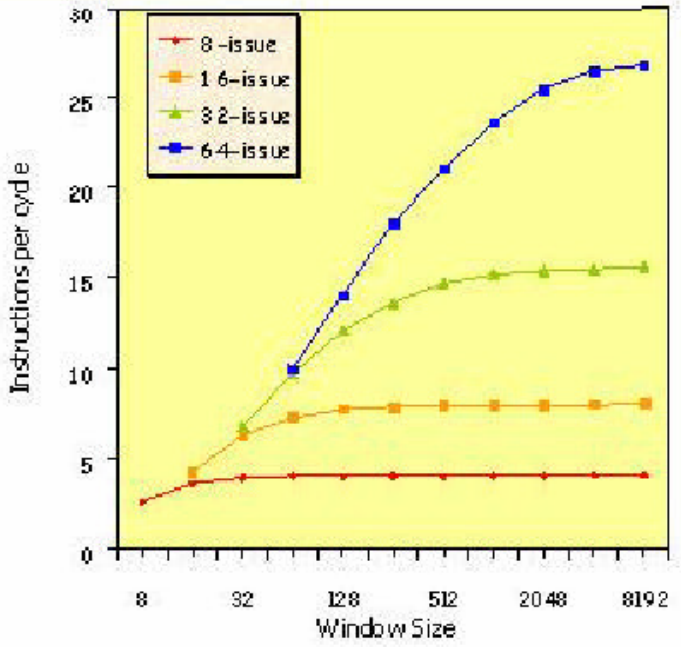
g. **Was versteht man unter Completion einer Instruktion?** [Wenn die FE die Asuführung beendet hat und das Ergebnis bereit steht zum zurückschreibn, dann heisst sie complete. Die Befehlsbeendigung findet in der Regel nicht in der Programmreihenfolge statt. Während der Beendigung wird die RS freigegeben und der Zustand der Ausführung wird im ReOrder Buffer abgelegt]

h. **Was versteht man unter dem Commitment?** [Nach der Beendigung werden die Befehle in ihre Programmreihenfolge zurücksortiert (commited). Durch das Zurücksortieren wird das Ergebnis endgültig in das Zielregister geschrieben]
    i. **Wann kann ein Befehl zurücksortiert werden?** [
    - wenn alle vorherigen Instrukktionen bereits zurücksortiert wurden oder im aktuellen Zyklus zurücksortiert werden können.]
    - wenn kein Interrupt vor der Instruktion stattgefunden hat
    - wenn sich der Befehl nicht mehr auf einem spekulativen Pfad (Vorhersage) befindet]

i. **Was versteht man unter Retirement?** [Ein Befehl geht in den Ruhestand (retires), wenn der entsprechende Eintrag im Reorder Buffer freigegeben wird. Dies passiert dann, wenn der Befehl entweder commited wurde oder abgebrochen!]

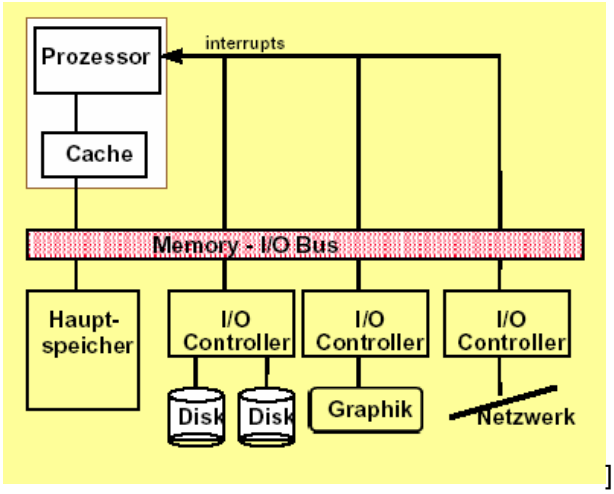
j. **Was unterscheidet einen Superskalarprozessor von einem VLIW Prozessor?** [dynamisch werden die Befehle vom linearen Instruction Stream aufgenommen und parallel berechnet. Beim VLIW haben wir eine großen, komplexen Befehl, der eine bestimmte Anzahl von Teil-Befehlen enthält, diese werden gleichzeitig geladen, dekodiert, zugewiesen und ausgeführt]

k. **Würde es Sinn machen die Windowsizes und die Zuordnungsbandbreit immer höher zu schrauben?** [Nein, das Verhältnis muss stimmen und irgendwann bringt eine größere Bandbreite einfach nichts mehr:



]

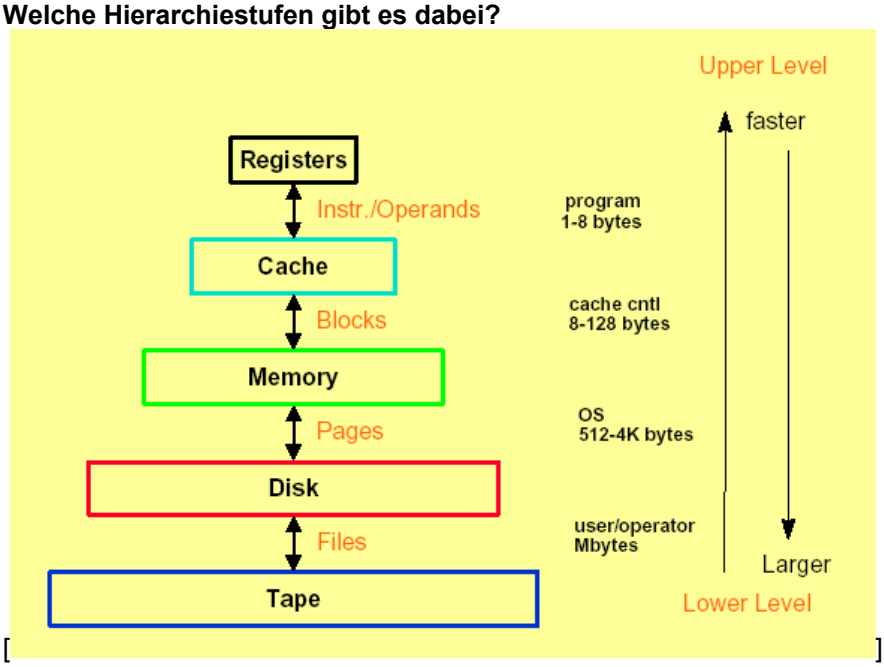
111. **Wie ist ein Rechner prinzipiell aufgebaut und welche Bestandteile hat er?** [Betandteile: Prozessor mit Cache | Hauptspeicher | Externe Speicher | Eingabegeräte | Ausgabegeräte | Busse. Aufbau:



]

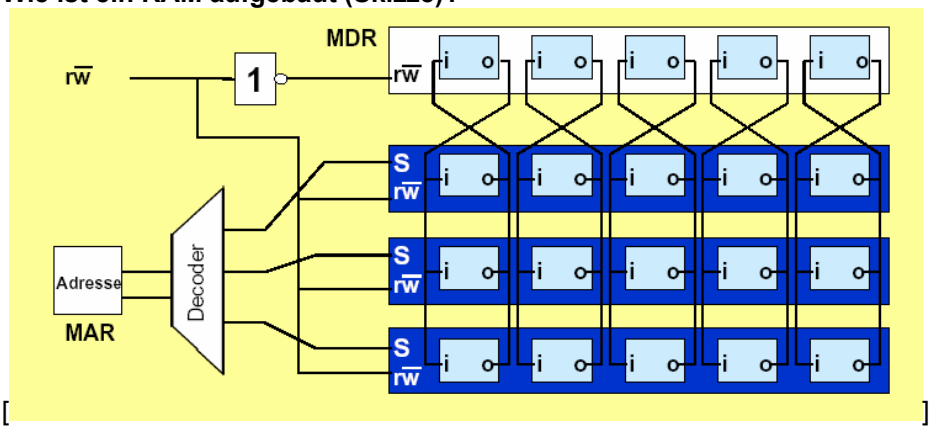
112. **Warum hat sich beim PC eine Speicherhierarchie entwickelt? Warum nimmt man nicht nur den superschnellen Cachespeicher?** [Cachespeicher ist sehr teuer. Hierarchie sinnvoll wegen Adressraum. Ein Register müsste z.B. eine sehr große Adresse ändern]

113. **Welche Hierarchiestufen gibt es dabei?**



[ ]

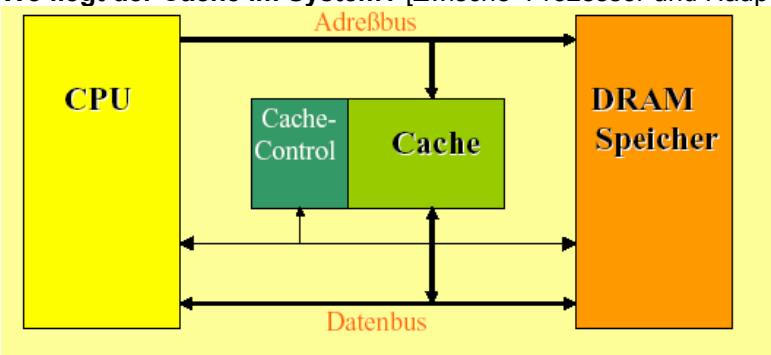
114. **Was versteht man unter Festwertspeicher?** [ROMs die vom Hersteller nach Maßgabe des Auftraggebers programmiert werden➔ EPROM kann gelöscht werden und wieder beschrieben. Oft gibt es Star- oder Initialisierungsprogramme (z.B. BIOS)]
115. **Was ist der Unterschied zwischen S-Ram und D-Ram?** [SRAM (Static Ram): Besteht aus regulären FlipFlops welches aus 6 Transistoren gebaut sind | DRAM: Nur ein Transistor. Die Infomation wird durch einen Kondensator gehalten und damit gespeichert ➔ Refresh notwendig. DRAM einige MBit Speicherkapazität. SRAMS sind viel kleiner]
116. **Wie ist ein RAM aufgebaut (Skizze)?**



[ ]

117. **Wieso ist ein Zugriff auf eine Hauptspeicherzelle langsamer als ein Zugriff auf ein Register?** [Hauptspeicher sind DRAM Zellen, Register sind aber SRAMs.. Bei einem Registerzugriff gibt es keinen Buszugriff]
    a. **Warum stellt man dem Prozessor statt dem Hauptspeicher nicht einige MByte Register zur Verfügung?** [Wäre sinnvoll, aber SRAMs sind schon physikalisch ca. 4 mal so groß wie DRAMS und ausserdem viel viel teurer! Aber die Entwicklung geht hin zu immer schnellerem Speicher bei kleineren Strukturen]
118. **Was ist ein Cache?** [frz. Cacher=verstecken. Er ist Softwaretransparent, d.h. der Benutzer und auch die Anwendungsprogramme bekommen nichts von seiner Existenz mit]
    a. **Kann man direkt auf eine Cachespeicherzelle zugreifen?** [nein]

b. **Wo liegt der Cache im System?** [Zwische Prozessor und Hauptspeicher:



Es gibt meist einen Instruktionscache und einen Datencache]

c. **Was sind die Ziele des Caches?** [Daten im Cache zu halten, die als nächstes gebraucht werden, damit der Prozessor diese aus dem Cache holen kann und nicht aus dem langsamen DRAM Speicher. Vorraussetzung. Auf bestimmte Daten wird häufig zugegriffen (z.B: in Schleifen) ➔ Lokalitätsprinzip]

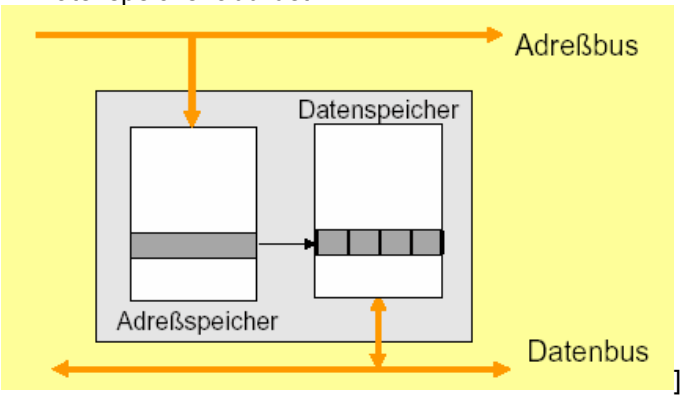
d. **Was versteht man unter einem Cache-Hit?** [gesuchtes Datum im Cache gefunden=> im gleichen Takt auslesen ohne langsamen Hauptspeicherzugriff. Überprüfung und Zugriff findet bei einem Cache-Hit in einem Zyklus statt]

e. **Was ist ein Cache-Miss und was passiert in diesem Fall?** [gesuchtes Datum im Cache nicht gefunden> ➔ Hauptspeicherzugriff ➔ übergeben an Cache und CPU! Geschieht Blockweise (d.h. auch umliegende Daten die höchstwahrscheinlich bald gebraucht werden➔ Wartezyklus bei Cache-Miss]

f. **Wie berechnet sich die mittlere Zugriffszeit beim Lesen eines Datums, wenn c die Zugriffzeit des Caches, m die Zugriffszeit beim Hauptspeicher und h die Trefferrate ist?** [c+(1-h)*m ➔ Auf den Cache wird immer zugegriffen und mit einer Wahrscheinlichkeit von 1-h müssen wir noch auf den Hauptspeicher zugreifen]

g. **Wenn der Cache voll ist müssen bestimmt Daten verdrängt werden, welche Strategien gibt es dazu?** [LRU➔Verdrängen des am längsten nicht benutzten Datums | LFU➔ Verdrängen des Datums auf das am wenigsten zugegriffen wurde | FIFO➔ Entfernen des Datums das am längsten im Cache ist (Schlange)]

h. **Wie ist der Cache aufgebaut?** [Der Cache ist bei kleinen Caches ein assoziativer (inhaltsorientierter Speicher, welcher eine Adresse im Adressspeicher auf ein Datum im Datenspeicher abbildet:

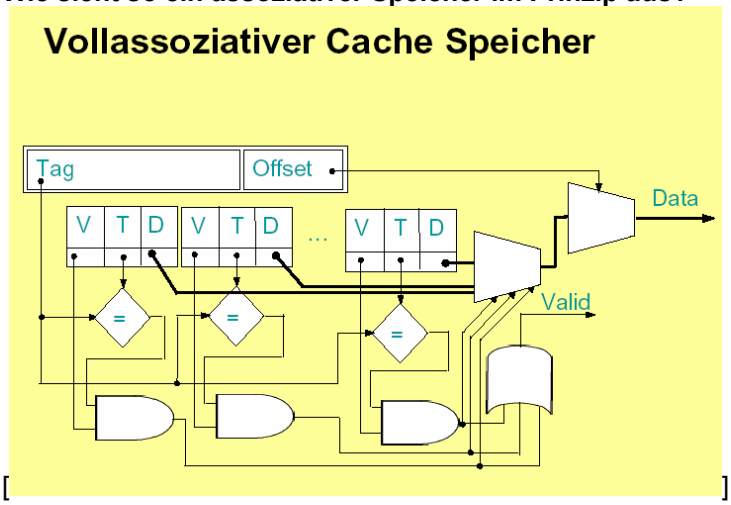


]

i. **Welche Informationen werden im Adressspeicher abgelegt?** [Adresse (Tag+Offset); Valid-Flag]

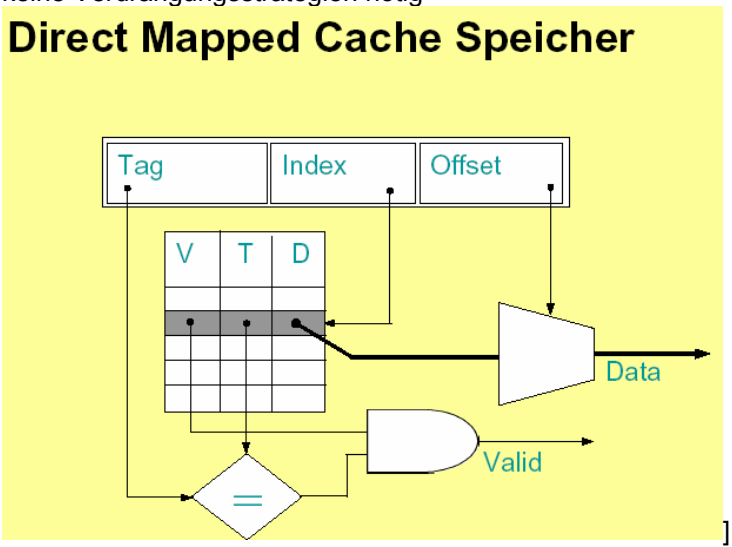
j. **Mit welcher Laufzeit (in O Notation) kann festgestellt werden ob ein gesuchtes Datum im Cache ist?** [Der Zugriff auf den Cache bzw. die Suche nach einer Adresse erfolgt parallel in O(1)]

k. **Wie sieht so ein assoziativer Speicher im Prinzip aus?**



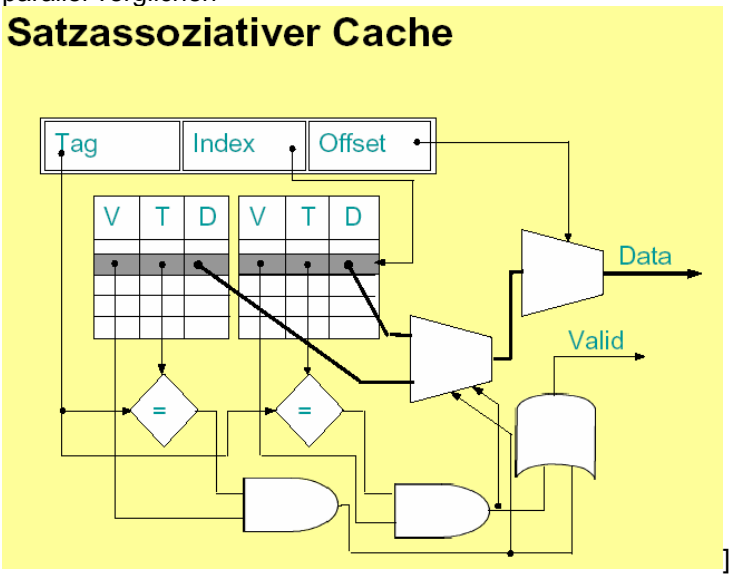
[ ]

l. **Was ist das besondere beim Direct Mapped Cache?** [Kein assoziativer Speicher nötig sondern feste Abbildung der Hauptspeicheradressen auf die Cache Adressen| keine Verdrängungsstrategien nötig



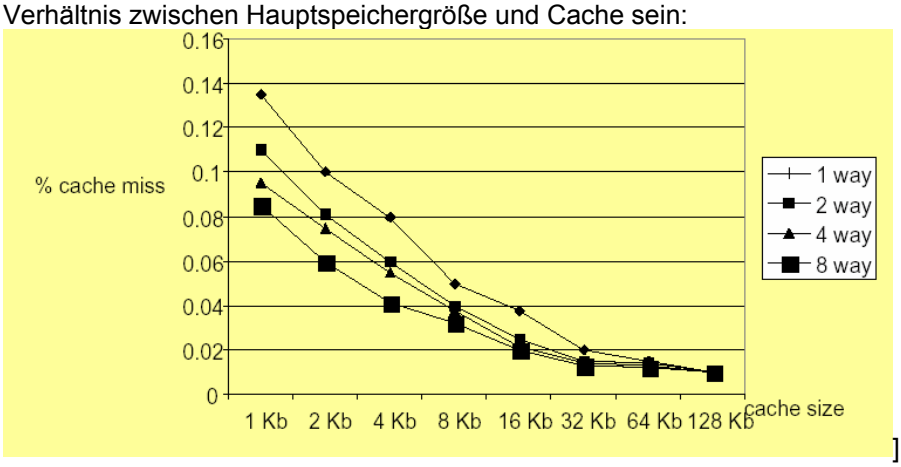
]

m. **Es gibt auch eine Mischung beider Cachevarianten, welche sich Satzassoziativer Cache nennt, was ist das besondere hieran?** [Index wählt einen Satz aus | Jeder Satz enthält mehrere Cache Lines | Tags in einem Satz werden parallel verglichen



]

n. **Macht es Sinn den Cache beliebig groß zu machen?** [Nein, denn die Cachemisses nehmen ab einer bestimmten Größe des Caches nicht mehr ab. Es muss ein gutes Verhältnis zwischen Hauptspeichergröße und Cache sein:



]

o. **Welche 3 Ursachen gibt es für Cache Misses?** [Compulsory➔Erster Zugriff erzeugt natürlich einen Miss
| Capacity➔ Programm braucht mehr Blöcke als in den Cache passen (größere Caches notwendig) | Conflict (bei Satzassoziativem Cache) ➔ Wenn der gemappte Satz bereits voll ist obwohl noch mehr Daten dazu kommen sollen (Lösung: Assoziativen Anteil erhöhen)]

p. **Nennen Sie die 3 verschiedenen Varianten von Schreibzugriffen auf den Cache?** [Write-Through, Write-Back, Write Allocation]
   i. **Wie geht write through bei einem Cache Miss vor?** [Datum wird nur in Hauptspeicher geschrieben und nicht in den Cache]
   ii. **Wie geht write through bei einem Cache Hit vor?** [auf beides wird geschrieben]
   iii. **Wie geht write back bei einem Cache Miss vor?** [Datum nur in Hauptspeicher, wie bei write through]
   iv. **Wie geht write back bei einem Cache Hit vor?** [schreiben nur auf Cache, setzen von Dirty bit. Hauptspeicher wird erst bei Verdrängung geschrieben]
   v. **Wie geht write allocation bei einem Cache Miss vor?** [schreiben nur auf Cache, setzen von Dirty bit. Hauptspeicher wird erst bei verdrängung geschrieben]
   vi. **Wie geht write allocation bei einem Cache Hit vor?** [schreiben nur auf Cache, setzen von Dirty bit. Hauptspeicher wird erst bei verdrängung geschrieben]

q. **Nennen Sie die Vor- und Nachteile von Write-Back/Write-Allocation gegenüber Write Through!** [Vorteil: Schnelle Schreibzugriffe, wenig Belastung des Systembusses| Nachteil: Konsistenz ist schwer]

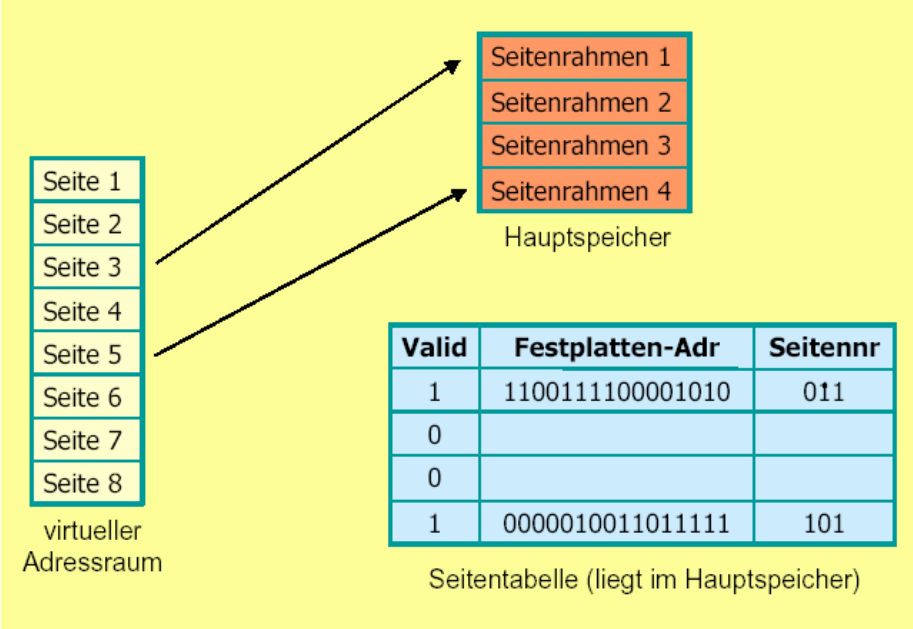
r. **Welchen Vorteil/Probleme hat die Verwendung eines Schreibpuffers beim Write-Back?** [Schneller, Problem: Konsistenz➔Puffer ist evtl. noch nicht komplett übertragen➔ Lösung: Warten bis Puffer leer ist, Inhaltsvergleich]

s. **Was versteht man unter „Early Restart" bzw. out-of-order-fetch?** [*Early Restart:* Wort wird beim eintreffen im Cache sofort an CPU weitergeben
Out-of-order-fetch: es wird zuerst das benötigte Wort geholt, dann der Rest aufgefüllt]

t. **Was bringt das hinzufügen einer 2. Cache-Stufe (2nd Level Cache bzw. 1st Level Cache)?** [Eine Verringerung der Zeit die benötigt wird bei einem Cache Miss das Datum aus dem Hauptspeicher zu laden. Die Wahrscheinlichkeit dass ein Datum in keinem der beiden Caches ist, ist entsprechend niedriger. Der Second Level Cache ist viel größer und natürlich auch etwas langsamer, aber immer noch um ein vielfaches Schneller als der Hauptspeicher. Es gilt im allgemeinen die Multilevel inclusion, d.h. alle Daten die im L1 stehen, stehen auch im L2]

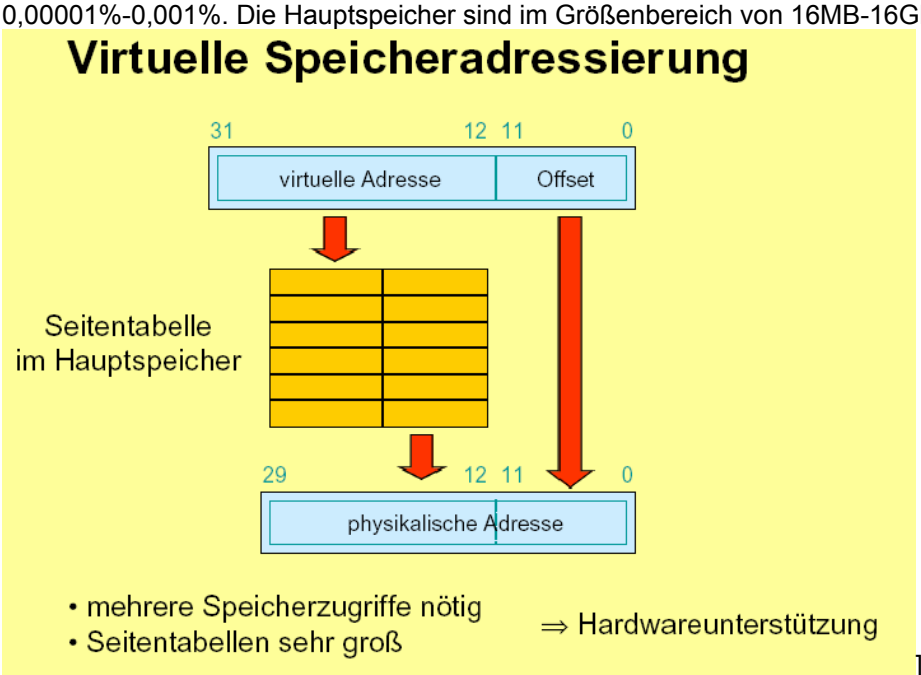
119. **Für was benötigt man virtuellen Hauptspeicher?** [32-Bit Programme gehen im allgemeinen davon aus, dass sie auch einen 32-Bit großen Speicherbereich zur Verfügung haben. Mit RAM ist das i.A. nicht zu machen, weshalb man die übrigen Adressen auf die Festplatte als virtuellen Speicher auslagert. Jedem laufenden Prozess steht dann neben dem Speicher im RAM auch ein Adressbereich im virtuellen Speicher zur Verfügung]

120. **Wie wird der virtuelle Hauptspeicher auf dem Sekundärspeicher verwaltet?** [vom Betriebssystem➔Welche Daten und Programmteile im RAM gehalten werden und welche Daten ausgelagert werden, bei vollem RAM]
    a. **Was ist Paging?** [Der Sekundärspeicher wir in Seiten (Pages) fester Größe unterteilt| Der Hauptspeicher besteht aus Seitenrahmen (PageFrames) die jeweils eine Seite aufnehmen können | Seitentabell (page table) speichert die Zuordnung



]

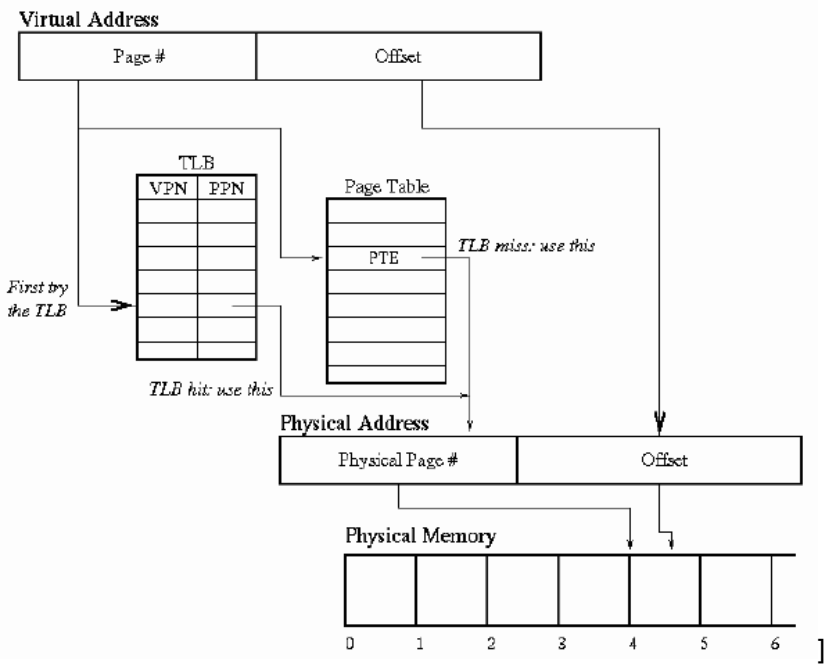
    b. **Nennen Sie typische Kenndaten des virtuellen Speichers!** [Die Seitengrößen sind zwischen 512 und 8192 Bytes. Ein Hit erfordert ca. 1-10 Taktzyklen ein Miss hingegen 100.000-600.000 Taktzyklen. Die Missrate liegt aber glücklicherweise bei nur 0,00001%-0,001%. Die Hauptspeicher sind im Größenbereich von 16MB-16GB



]

121. **Was wird im Translation Lookaside Buffer abgelegt?** [die zuletzt berechneten Abbildungen von virtuellen auf physikalische Adressen, also so etwas wie ein Adresscache

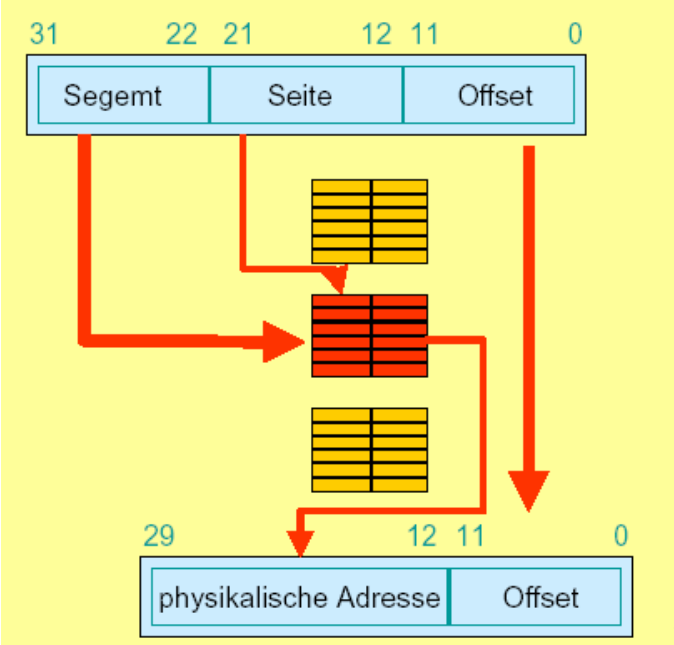
für die Adressabbildungen.



]

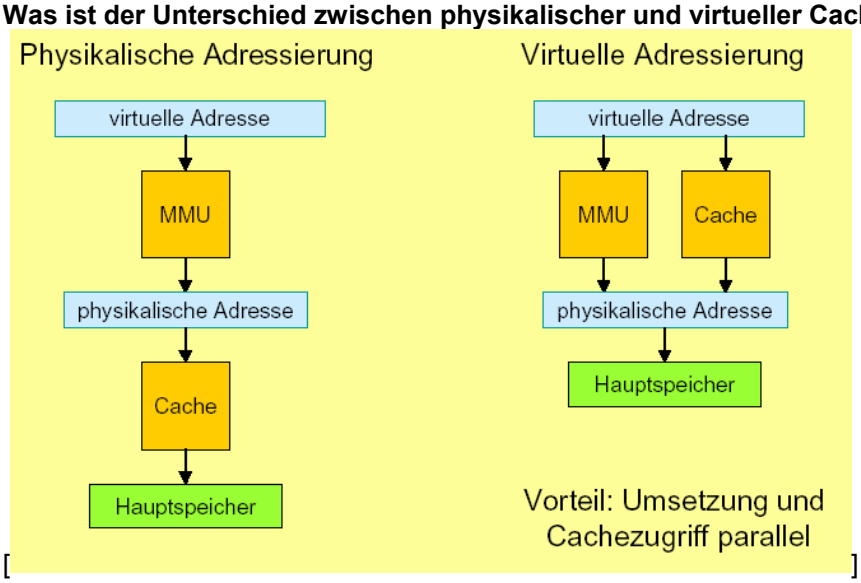
122. **Wie würde ein optimaler Seitenersetzungsalgorithmus lauten?** [Entfernen nach maximaler Anzahl vergangener Instruktionen]
123. **Leider ist dieser ab nicht zu implementieren, welche Strategien kann man anstelle dessen verwenden?** [LFU/FIFO/LRU]
    a. **Wie funktioniert LFU?** [Jede Seite hat einen Zähler, der bei jedem Schreib- oder Lesezugriff inkrementiert wird. Bei Verdrängung wird diejenige Seite entfernt, die den kleinsten Zugriffszählerwert aufweist! Das Alter der Seite wird NICHT berücksichtigt➔Frische Seiten haben schlechte Karten]
    b. **Wie funktioniert FIFO?** [Prinzip der Warteschlange. Die jeweils älteste Seite wir entfernt]
        i. **Bei dem FIFO Algorithmus gibt es eine gute Variante, die der Seitenersetzung eher gerecht wird, falls kurz vor dem rausfliegen einer Seite ein Zugriff auf diese erfolgt. Welcher „Trick“ wird hierbei angewandt?** [R-Bit wird gesetzt, falls ein Zugriff auf die Seite stattfindet => Second Chance Ersetzung, das heisst die Seite darf sich noch mal am Ende Anstellen, das R-Bit wird dann auf NULL gesetzt. Somit wird aus der Schlange ein „Ring mit Ziellinie“]
    c. **Wie funktioniert LRU?** [Die Seite, die am längsten nicht mehr benutz wurde, wird ersetzt. Seitenrahmen erhalten Timestamp]
    d. **Neben dem einfachen LFU Algorithmus gibt es noch eine Variante namens NRU (Not Recently Used). Welche Strategie wird hierbei angewandt?** [Es gibt 2 Bits, ein R-Bit (Referenzbit) und ein M-Bit(Modification ist LSB) ➔ 4 verschieden Kombinationen ergeben 4 Klassen ➔ Seite aus der niedrigsten nichtleeren Klasse wird ersetzt. Periodisches Löschen des R-Bit]
124. **Was bedeutet Segmentierung des Hauptspeichers?** [Einteilung in verschiedene unterschiedlich große Bereiche zB für Programmcode, Stack, statische Variablen, Prozesse; Die Segmente können durch unterschiedliche Zugriffsrechte geschützt werden. Die Verdrängung kann für bestimmte Segmente verhindert werden, wenn diese eigentlich permanent im Hauptspeicher gehalten werden sollen (z.B. Programmcode)]
    a. **Paging und Segmentierung kann man sinnvollerweise auch kombinieren, wie geht das?** [Jedes einzelne Segment wird unabhängig von den anderen mit Paging

realisiert



]

125. **Was ist der Unterschied zwischen physikalischer und virtueller Cacheadressierung?**

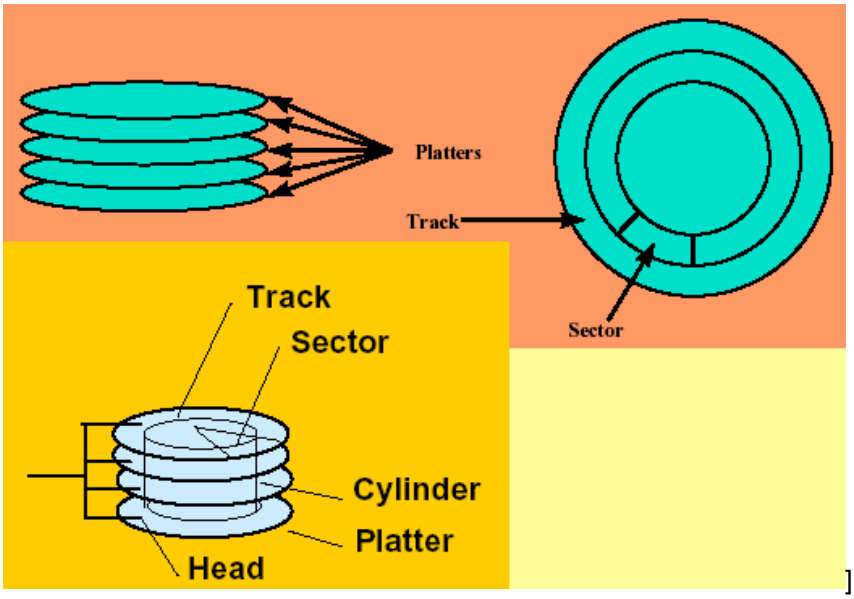


[ ]

a. **Welche Probleme treten bei der virtuelle Cachadressierung auf?** [Bei jedem Prozesswechsel muss der Cache geleert werden für den neuen Prozess | „synonym Problem“ ➔ Gleiche virtuelle Adresse durch verschiedene Prozesse genutzt für verschiedene reale Adressen]
b. **Wie könnte man die Probleme die bei der virtuellen Cacheadressierung auftreten verbessern?** [PID Process identifier Tag, welches die virtuelle Adresse eindeutig macht]
c. **Was ist der entscheidende Vorteil der physikalischen Adressierung?** [Cache muss nicht bei jedem Prozesswechsel geleert werden]

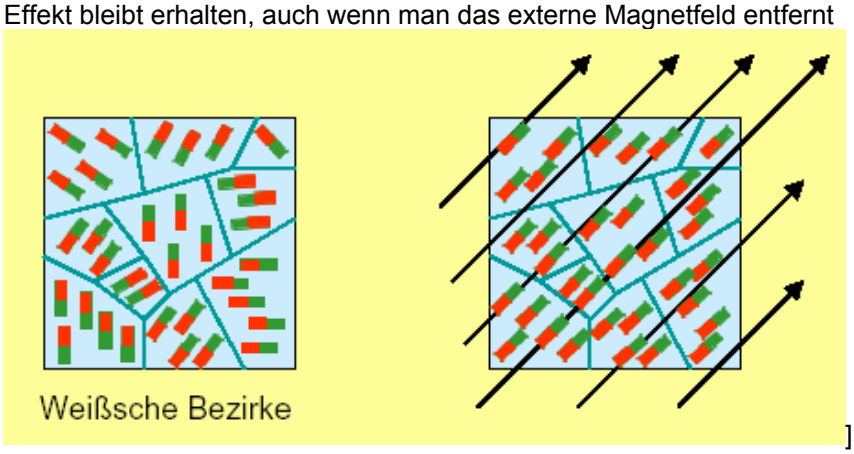
126. **Erklären Sie die folgenden Begriffe bei einer Festplatte: Platte, Spur, Sektor, Cylinder, Head!** [
Platte: Die einzelnen physischen Platten in einer Festplatte ➔ Typischerweise 1-8 Platten. Eine Platte besteht aus nichtzusammenhängenden konzentrischen Kreisen
Spur (Track): Ein kompletter Kreis auf einer Platte
Sektor: Kreisabschnitte einer Platte in einem bestimmten bereich. Ein Sektor ist die kleinste beschreibbare Einheit
Cylinder: alle übereinander liegenden Tracks von der untersten zur obersten Platte
Head: Der Schreib/Lesekopf ➔ Typischerweise 2-16 Stück (2 pro Platte, einer unten, einer oben). Eine Festplatte rotiert typischerweise mit 3600 bis 15000 Umdrehungen in der

Minute



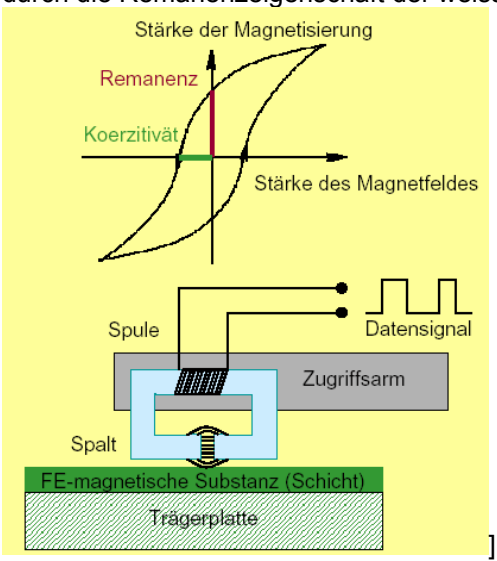
]

127. **Welche Form von magnetismus wird zur Speicherung von Daten auf Festplatten genutzt?** [Ferromagnetismus ➔ Verstärkung des externen Magnetfeldes➔ Remanenz: Effekt bleibt erhalten, auch wenn man das externe Magnetfeld entfernt



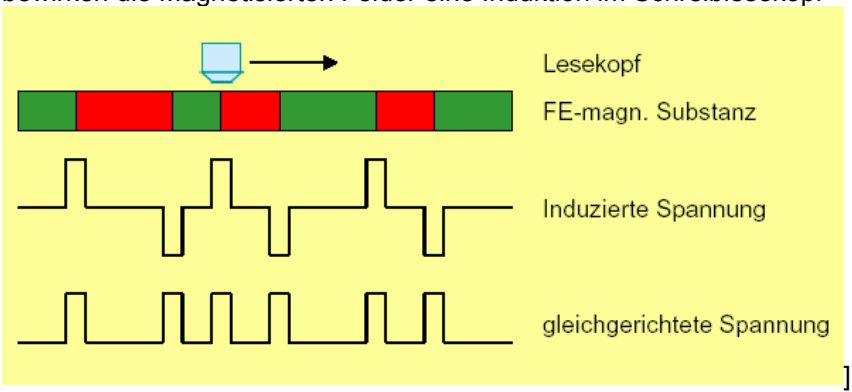
]

a. **Wie wird magnetisiert, d.h. geschrieben?** [Es wird ein externes elektromagnetisches Magnetfeld angelegt um die weisschen Bezirke auszurichten (d.h. zu magnetisieren). Die Magnetisierung hat hysteresverhalten. Die Daten werden durch die Remanenzeigenschaft der weisschen Bezirke gehalten



]

b. **Wie wird gelesen?** [Durch Induktion ➔ Wird der Lesekopf über die Platte beweg so bewirken die Magnetisierten Felder eine Induktion im Schreiblesekopf

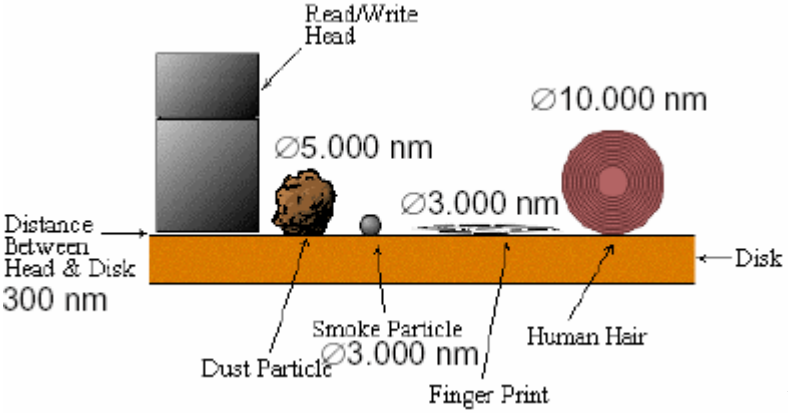


]

c. **Welche Arte von Magnetismus gibt es denn sonst noch?** [Diamagnetismus, Paramagnetismus]

d. **Welche Arten von Schreibleseköpfen gibt es?** [Ferrit Köpfe ➔ Spule mit Ferritkern. Die Größe ist nach untern begrenzt. Sie sind recht schwer
Dünnfilm Köpfe➔Aus der Halbleitetechnik, leicht und klein
MR-Köpfe➔Magnetoresistive Köpfe sind empfindlcher, aber dafür leichter, kleiner und ermöglichen eine höhere Speicherdichte]

e. **Wie hat man sich das mit dem Schreiblesekopf über den Platten vorzustellen?** [Der Schreiblesekopf fliegt durch den Luftstrom getragen über die Platte. Bei Änderung des Luftdruckes ändert sich der Abstand. Es gibt eine Parkspur in welcher er landet beim Ausschalten der Platte, in welcher natürlich keine Daten stehen dürfen



]

f. **Wie werden die Daten auf dem Datenträger Kodiert?** [Es gibt 3 Verfahren: FM Kodierung: Immer eine Transition am Anfang der Periode (Taktimpuls) – Bei 1 Bits eine Transition in der Mitte | MFM geht sparsamer mit den Takimpulsen um und setzt diese nur, wenn im vorherigen und im aktuellen Periode kein Impuls stattfindet ➔ Doppelte Datendichte | RLL Kodierung: Kein Taktsignal! Anzahl der konsekutiven 0-Bits ist beschränkt. 1-Bit ist immer umgeben von 0-Bits. Synchronisierung nicht durch Taktsignal sondern durch Laufzei zwischen den 1-Bits ➔ Aufwendige Codierung aber 3x mehr Daten als bei MFM!!!!]

128. **Wie läuft ein typischer Festplattenzugriff ab?** [Beweg Kopf zum richtigen Zylinder, Warte bis der richtige Sektor zum Kopf rotiert ist, dann Übertragung]

129. **Mit welchen Strategien werden mehrere Anfragen an die Festplatte abgearbeitet?** [FCFS|SSF|Fahrstuhlalgorithmus]

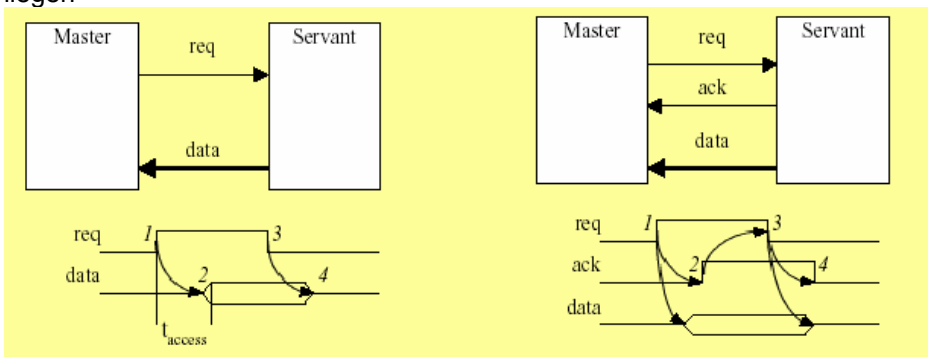
a. **Wie funktioniert FCFS (First-Come-First-Served)?** [Anfragen warden der Reihe nach abgearbeitet ➔ u.U. viele Lesekopfbewegungen]

b. **Wie funktioniert SSF (Shortest-Seek-First)?** [Das Element zu welchem der Abstand des SL-Kopfes gerade am geringsten ist wird genommen]

c. **Wie funktioniert der Fahrstuhlalgorithmus?** [Es wird in einem Durchgang entweder hoch oder runter gefahren und nicht zwischendrin gewechselt]

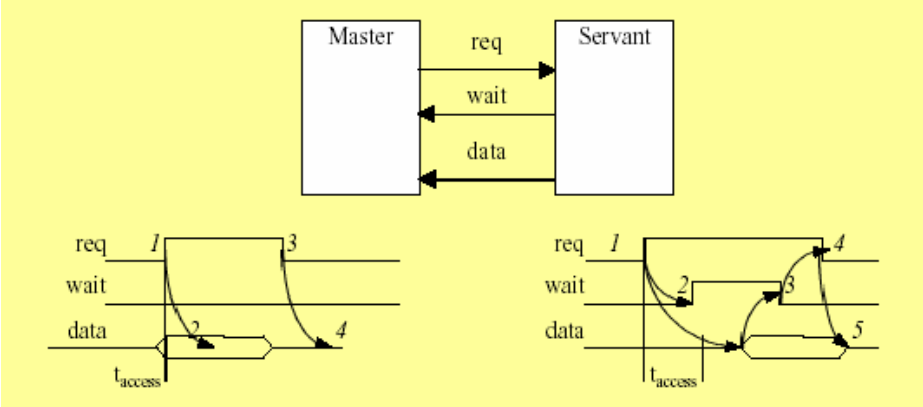
d. **Was ist der Unterschied zwischen SSF und dem Fahrstuhlalgorithmus?** [Kopfbewegung zu nächsten vs. Auf und absteigende von links nach recht und umgekehrt]

130. **Wie sieht eine MS-DOS Festplatte nach der Formatierung aus?** [
Systembereich bestehend aus:
- Urladerbereich, d.h. Bootblock➔Kurzes Programm zum Aktivieren des Startens des Bestriebsystems in den Speicher(<512Bytes)
- Dateizuordnungstabelle (File Allocation Block)➔
- Dateiverzeichnis (directory
Datenbereich]
131. **Welche unterschiedlichen Lösungen gibt es zur Speicherung von Dateien auf einer Festplatte, was sind die Vor und Nachteile?** [Kontinuierliche Allokation=> Hohe Performance, aber Fragmentierung| Allokation mittels verknüpfter Liste=> Keine Fragmentierung, aber Zugriff länger]
132. **Wie funktioniert das FAT Verfahren?** [Kombination aus Index + verkettete Liste➔Nachteil: Einteilung in Blöcke fester Größe die verkettet sind. Ein Block kann immer nur eine Datei bzw. Teil einer Datei enthalten. Die FAT benötigt sehr viel Speicher im RAM z.B. bei 512 MB Platte mit 1Kbyte Blockgröße sind das 2MB!!!]
133. **Eine andere Strategie als die FAT sind die I-Nodes, wie funktioniert dieses Verfahren?** [Zu jeder Datei wird eine kleine Tabelle (I-Node) verwalten, die die typischen Dateiinformationen wie Größe, Erstellungsdatum Modifikationsdatum, Verweise speichert]
134. **Was versteht man unter Asynchroner bzw. Synchroner Kommunikation?** [Asynchron: Speicherung über einen Puffer | Synchron: Direkte Kommunikation]
135. **Was ist ein Interface?** [Datenpuffer bei unterschiedlichen Übertragungsraten, Synchronisation und Datenkonvertierung, z.B. D/A, A/D, Seriell nach Parallel etc.]
136. **Was ist ein Bus?** [Ein Bündel von Drähten mit festgelegter Funktion, d.h. z.B. Adressbus oder Datenbus]
137. **Was ist ein Port?** [Er verbindet das System mit den Bussen und damit mit der Umwelt]
138. Wie kann man den Zugriff auf den Kommunikationsbus steuern? [Arbiter, Prioritäten, Daisy-Chain ähnlich Token Ring beim Netzwerk]
139. **Wie wird in der Regel durch das Interface-Protokoll geschrieben bzw. gelesen?** [Lesen: Die zu lesende Adresse wird auf die Adressleitung gelegt, dann wird das „enabled“ Signal gesetzt. Jetzt können die Daten gelesen werden und werden auf die Datenleitung gelegt. Nach dem Lesen wird die enabled Leitung wieder auf Low gesetzt
Schreiben: Setzen des Schreibsignals, gleichzeitig anlegen der zu schreibenden Adresse und der in diese Adresse zu schreibenden Daten. Danach (nach Setup-Zeit) kann die „enabled“ Leitung wieder aktiv werden und die Daten werden geschrieben]
    a. **Welche Grundlegenden Methoden gibt es dabei?** [Strobe Protokoll: Master macht ein Request geltend um Daten zu anzufordern. Der Servant legt die Daten auf den Bus innerhalb einer gewissen Zeit (t_{access}). Dann empfängt der Master die Daten und zieht sein Request zurück. Der Servant wartet nun auf den nächsten Request.
    Handshake Protokoll: Genau gleich wie Strobe Protokoll nur dass der Servant nach dem versenden der Daten ein ACK setzt (d.h. dass die Daten sicher auf dem Bus liegen



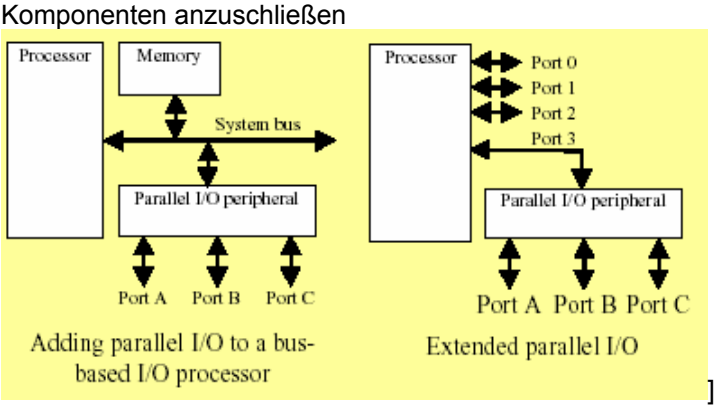
    Strobe Protokoll Handshake Protokoll]
    b. **Es gibt auch einen Kompromiß der beiden Protokolle, welcher ohne ACKL auskommt, aber dennoch im Erfolgsfall die Daten so schnell sendet wie das Strobe Protokoll. Wie geht das?** [Kann der Servant die Daten innerhalb der t_{Access} Zeit zur Verfügung stellen, dann entspricht diese dem Strobe Protokoll´(Diesen Fall nennt man „Fast-Response“. Falls der Servant nicht schnell genug reagieren kann, setzt er ein „Wait“ Signal“ bis er die Daten zur Verfügung hat und auf den Bus legen kann, dann wird das Wait Signal wieder (Diesen Fall nennt man Slow-Response).

zurückgesetzt



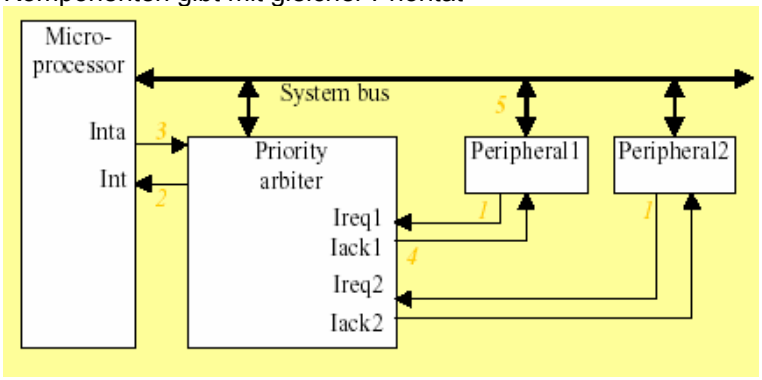
Diese Technik findet z.B. beim ISA Bus Anwendung]

140. **Welche beiden Varianten der E/A Adressierung gibt es?** [
Port-Base I/O➔Prozessor hat mehrere N-Bits-Ports| Der Softwarezgriff auf Ports erfolgt über spezielle Register/Speicherbereiche oder spezielle I/O-Befehle. Ein Port spezifiziert direkt eine externe Komponente
Bus-Based I/O➔Prozessor hat Adress-, Daten- und Kontrollport, die zusammen einen Bus bilden. Das Protokoll ist im Prozessor integriert. Eine einfache Operation führt das Schreib-/Leseprotokoll auf dem Bus durch. Die Empfängeradresse muss angegeben werden]
    a. **Was sind die Probleme, wenn der Prozessor auf das eine oder andere System festgelegt ist?** [Port-Based I/O: Evtl. zuwenig Port für mehrer parallel zu betreibende Geräte | Bus-Based I/O: Der Prozessor unterstützt keine parallele Kommunikatioen, d.h. man benötigt zusätzlich etwas, dass einem ermöglicht parallel arbeitende Komponenten anzuschließen



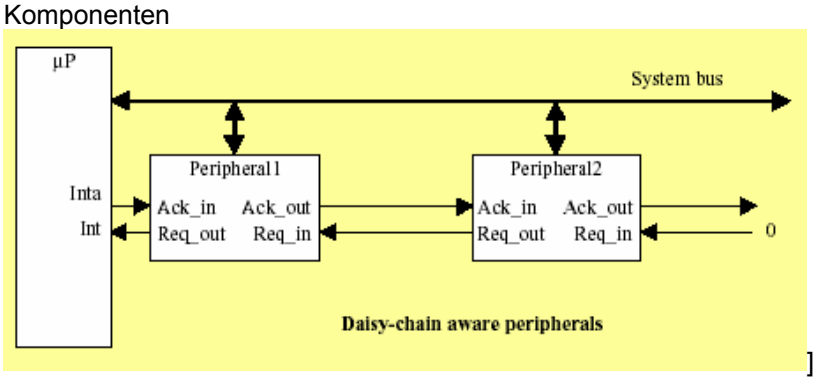
]

141. **Wie löst man das Problem, wenn mehrer Komponenten gleichzeitig den Bus benutzen wollen?** [Es gibt einen Priority Arbiter, der den Komponenten den exklusiven Zugriff erteilt. Die Komponenten haben zusätzlich Prioritäten. Es gibt feste Prioritäten, welche vergeben werden, wenn man den Komponenten eine eindeutige Rangordnung zuweisen kann, und es gibt Rotierende Prioritäten (Round-Robin)➔ die höchste Priorität (also ein Token) wird der Reihe nach weitergegeben. Dies ist gerechter, wenn es einige Komponenten gibt mit gleicher Priorität



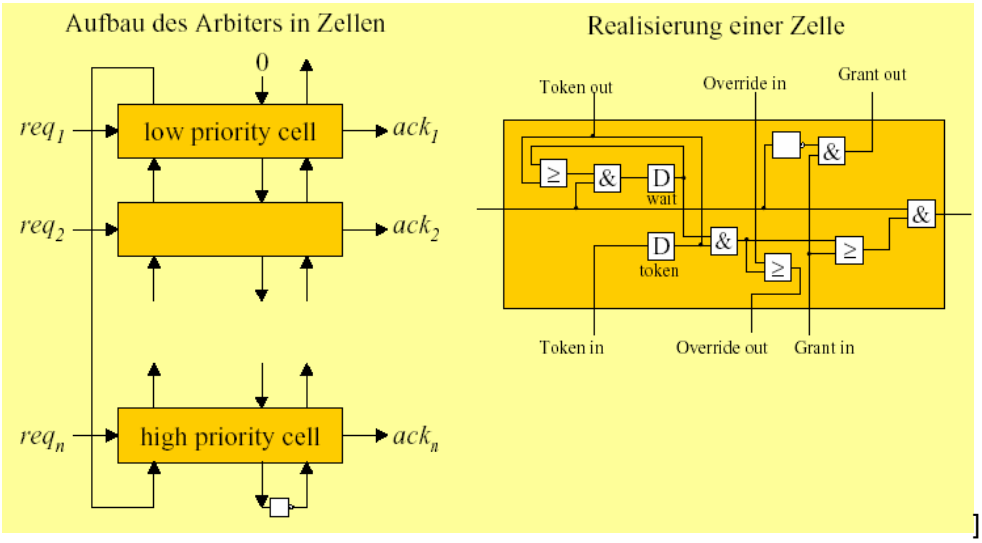
142. **Was versteht man unter Daisy-Chain Arbitrierung?** [Es gibt keinen Arbiter, sondern jede Komponente hat einen Req Eingang und einen Ack Ausgang. Nur eine der Komponenten ist mit der Ressource verbunden. Req fließt in Richtung Ressource, Ack fliest in Richtung der

Komponenten



]

a. **Es gibt eine Variante die zusätzlich zur Daisy-Chain einen Arbiter mit Priority und Round Robin. Was ist hier anders?** [Die Komponenten hängen mit ihren Req und ACK Leitungen direkt am Arbiter, dieser Arbeitet die Komponenten aber trotzdem der Reihe nach ab (Daisy Chain mit Round Robin). Vorteil: Wenn ein Gerät ausfällt bricht nicht alles zusammen

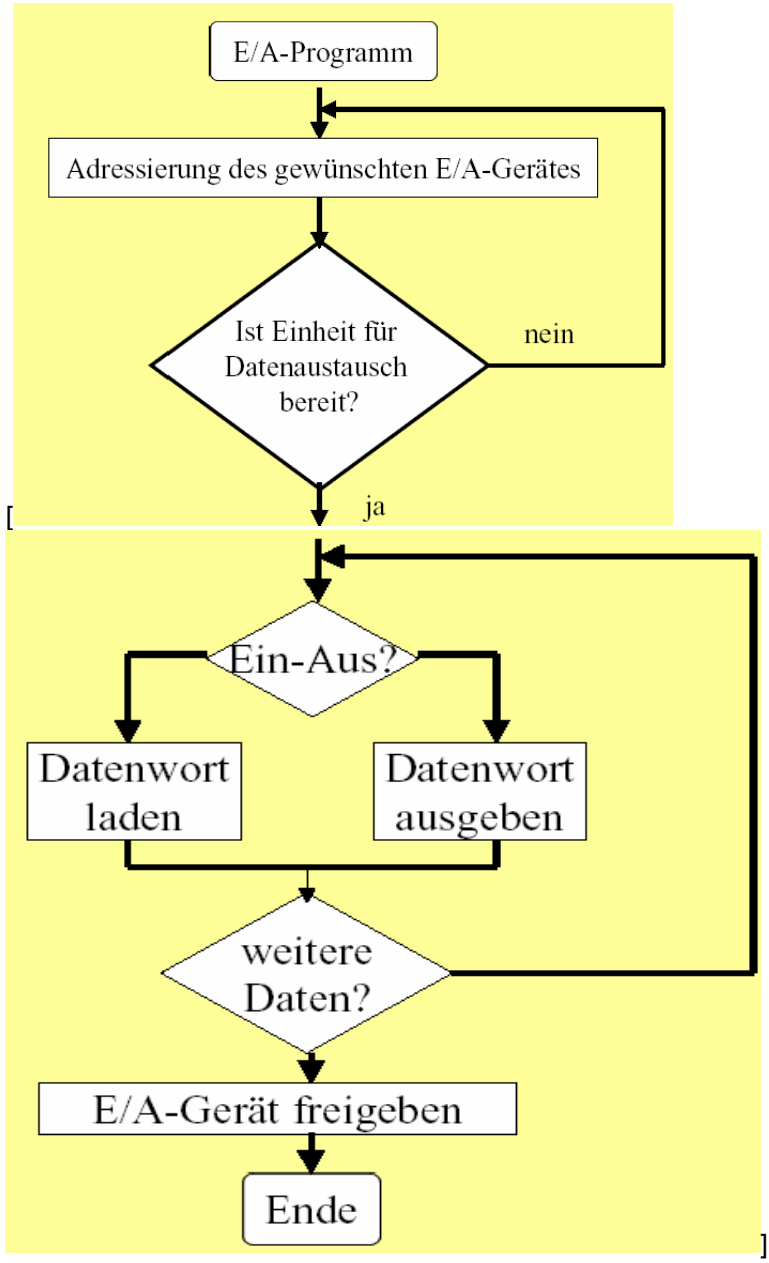


]

143. **Welche Vor- und Nachteile bringt eine Ringförmige Verwaltung des Busses?** [Wie bei Token Ring➔ Neue Komponenten sind einfach zu implementieren, aber eine defekte Komponente legt alle anderen lahm ➔ Lösung mit extra Arbiter der die Komponenten Sternförmig verbindet]
144. **Welche 3 Möglichkeiten gibt es den Prozessor mit der Peripherie kommunizieren zu lassen?** [Programmierte E/A, das heisst ein Programm steuert die Daten |
Interrupt: Externes Signal erzwingt Unterbrechng des laufenden Programms, Die Datenübertragungs wird durch speziell Routine erledigt |
DMA(=Direct Memory Access): Durch eine DMA Prozessor wird ein separater Datenweg direkt zwischen Speicher und Peripherie geschaffen➔ Die eigentliche CPU wird dadurch entlastet]
    a. **Was versteht man unter Memory Mapped I/O?** [Der I/O Port hat eine eindeutige Adresse, d.h. der Zugriff kann stattfinden wie beim Zugriff auf eine normale Speicherzelle➔Die Adresse existiert so im RAM oder ROM nicht!
    b. **Was versteht man unter I/O Mapped?** [Der Prozessor verfügt über spezielle I/O Befehle wie IN <addr> oder OUT <addr>. Die I/O Geräte haben einen eigenen Adressbereich und eigene Steuersignale zur Unterscheidung. Der I/O Port ist direkt mit dem dem Steuerbus verbunden. Es werden zusätzlich zu den Daten ständig synchronisationssignale geschickt.]
145. **Wie Synchronisieren sich zwei Kommunikationspartner?** [
Strobing(Anstoßverfahren➔Empfänger immer Empfangsbereit und schneller als Sender Unidirektionale Übertragung. Der Sender verlässt sich darauf, dass der Empfänger immer Empfangsbereit ist | Polling(Statusabstimmung)➔Empfangsbereitschaft wird über Pollingsignal geprüft, Sender schneller als der Empfänger. Das Pollingsignal muss permanent abgefragt werden (Busy Waiting)|
Handshaking(Gegenseitige Abstimmung)➔Hohe Übertragungssicherheit, Synchronisation

von Sender und Empfänger mit Polling und Strobing. Der Sender wartet bis der Empfänger bereit ist, dann wartet der Empfänger auf den Sender. Nun erfolgt der Datenaustausch]

146. **Wie läuft der Datenverkehr schematisch ab?**

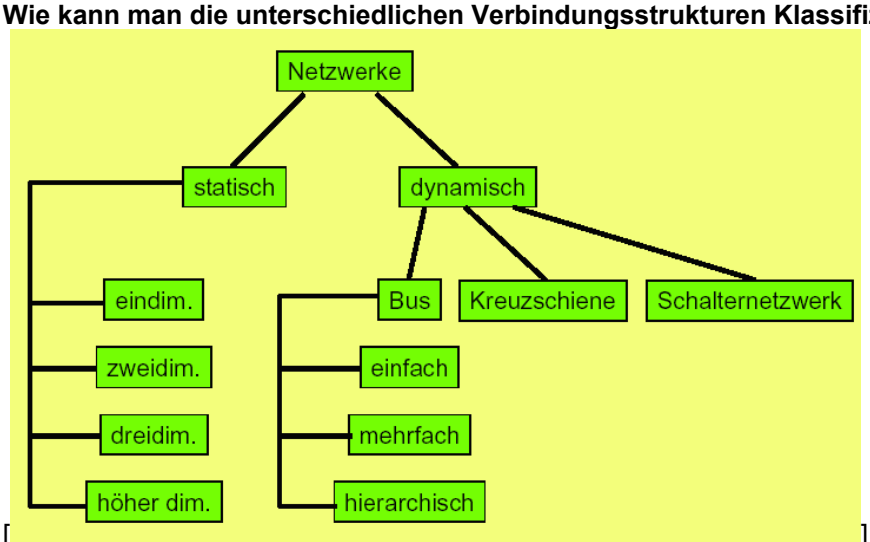
[

]

147. **Das versteht man unter DMA Zugriff?** [Direkter Zugriff einer Komponente auf den Speicher über einen speziellen DMA Channel ohne den Prozessor anzusprechen. Der Speicher ist dann ein sogenannte Zweitorspeicher, d.h. der Speicher kann einerseits direkt über den Prozessor] angesprochen werden und zweitens über den DMA-Channel➔Problem: Sehr teuer und Inkonsistenzen bei gleichzeitigem Zugriff über beide Tore möglich]
    a. **Was ist dabei Cycle Stealing** [Prozessor wird beim DMA Zugriff angehalten (meist über IRQ)]
    b. **Was ist Memory Idle?** [Bei der Befehlsausführung gibt es ja zwei Phasen, nämlich das Befehl holen und das Befehl Ausführen. In der Zweiten phase nutzt der Prozessor den Speicherzugriff nicht, das heisst der Speicher ist im Idle Modus. Diesen Zeitpunkt nutzt man für DMA Zugriffe. Die DMA Einheit erhält ein Memory Idle Signal vom Prozessor]
148. **Wie wurde historisch gesehen die Performance der Rechner verbessert?** [Bis 1985 gab es nur Parallelität auf Bitebene, das heisst z.B. Multiplizierer und Addierer auf der ALU konnten parallel betrieben werden | Ab 1985 kamen erste Konzepte wie Pipelining hinzu und mehrer Funktionseinheiten(Superskalare Prozessoren). Die Parallelität erweiterte sich

damit auf die Instruktionsebene | Seit 1990 werden zusätzlich zu dem eigentlichen Prozessor noch Caches und Hauptspeicher auf der Chipfläche untergebracht | Inzwische n gibt es natürlich auch parallelität auf Prozessoreben (Multiprozessorsysteme)]

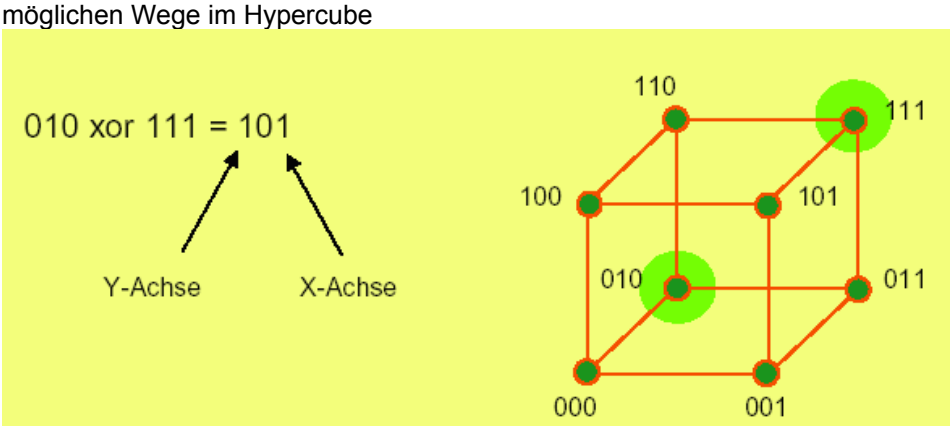
149. **Welche verschiedenen Parallelrechnermodelle gibt es die durch Cache, Prozessoren und Hauptspeicher klassifiziert werden?** [Verschiedene Kombinationen:
Homogenität der Prozessoren➔ Homogene vs. Heterogene Parallelrechner
Hierarchie der Prozessoren➔Symmetrische Parallelrechner ➔ alle gleichberechtigt und damit austauschbar | nichtsymmetrische Parallelrechner➔ Es gibt Masters und Slaves
Eigenständigkeit der Prozessoren➔ lose gekoppelte Parallelrechner➔Netzwerk von eigenständigen Rechnern | eng gekoppelte Parallelrechner ➔ Physikalisch ein Rechner
Verwendeter Speicher➔ Verteilter Speicher mit getrennten Adressräumen | Verteilter Speicher mit gemeinsamem Adressraum | Nicht verteilter Speicher mit gemeinsamem Adressraum]
150. **In welche Klassen teilte Flynn 1966 verschieden Rechnermodelle ein?** [SISD,SIMD,MISD,MIMD]
151. **Durch welche Charakteristiken zeichnet sich eine Verbindungsstruktur aus?** [*Komplexität (Kosten*), der Verbindungsgrad (Grad eines Knotens),
Diameter (Maximaler Pfad zwischen zwei Knoten,
Regelmäßigekeit (Je regelmäßiger desto einfacher zu implementieren),
Notwendige Leitungslängen (je Kürzer desto besser),
Blockierung (Es sollte immer alernativpfade geben, damit eine bestehende Verbindung eine andere nicht blockiert, d.h. ein Pfad schon belegt ist
Erweiterbarkeit: Die Anzahl der möglichen Prozessoren kann begrenzt, stufenlos erweiterbar oder nur durch verdopplung der Anzahl erweiterbar sein
Skalierbarkeit: Durch die Eweriterung sollen die wesentliche Eigenschaften des Verbindungsnetzes beibehalten werden können
Ausfalltoleranz durch Redundanz: Auch beim Ausfall svon Teilen des System soll die Gesamtfunktionalität erhaltgen bleiben (Graceful Degradation)
Durchsatz (Übertragungsbandbreite): Die maximale Übertragungsrate
Komplexität der Pfadberechnung: Die Berechnung des Pfades von einem zum anderen Bauteil sollte möglichst einfach sein (schneller Hardwarealgorithmus)]
152. **Was unterscheidet ein statische Netz von einem dynamischen?** [Beim statischen Netz sind die Vebrindungen fest verdrahtet | Beim dynamischen Netz gibt es Schaltnetzkomponenten, die bestimmte Eingänge auf bestimmte Ausgänge durchschalten (d.h. vermitteln)➔ Fest installierte Verbindungen gibt es nicht]
153. **Wie kann man die unterschiedlichen Verbindungsstrukturen Klassifizieren?**



[ ]

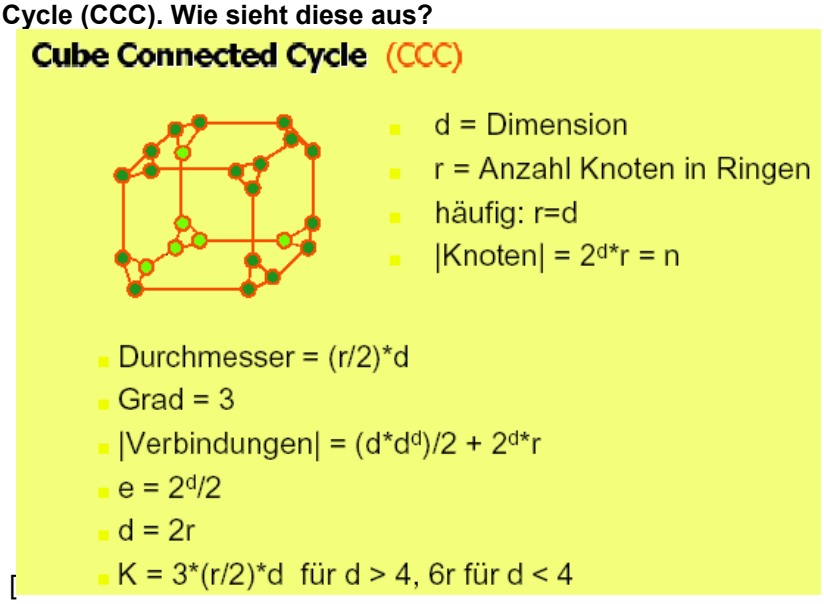
a. **Nenne Sie hierzu jeweils ein Beispiel!** [1-Dimensional➔ Kette |
2-Dimensional➔Vollständiger Verbindunggraph (Problem: Riesiger Fabout➔teuer)|
Stern(Problem: Zentraler Knoten ist Flaschenhals)|Ring|Gitter|Mesh
3-Dimensional: Hypercube (d-dimensionaler Würfel)| Cube Connected Cycle]
b. **Was versteht man beim Hypercube unter eCube Routing?** [alle Knoten werden binär kodiert. Verknüpft man nun Start- und Zielknoten mit XOR, dann erhält man die

möglichen Wege im Hypercube



]

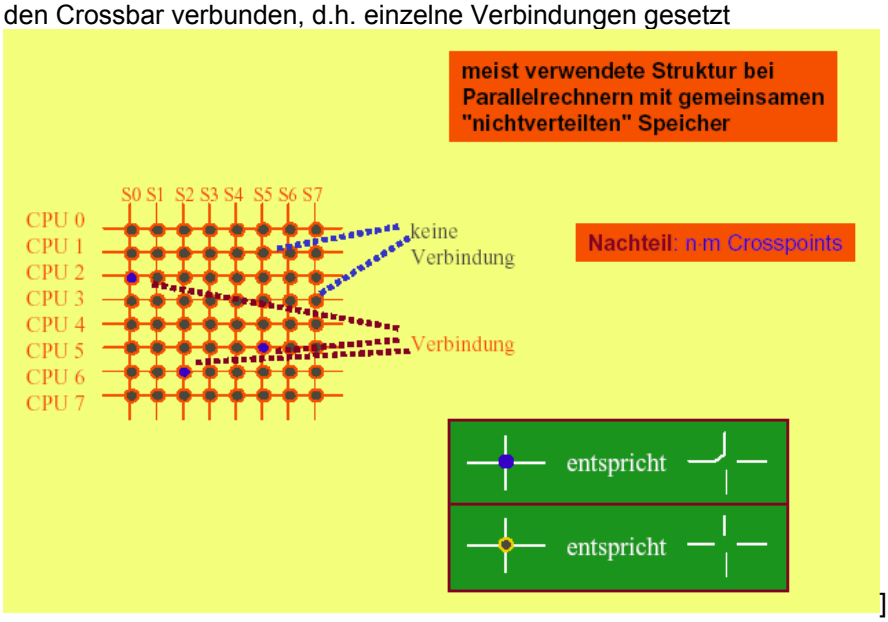
154. **Bei den statischen Verbindungsstrukturen gibt es eine sehr interessante Struktur die sonst eher nicht erwähnt wurde (in anderen Vorlesungen) sie heisst Cube Connected Cycle (CCC). Wie sieht diese aus?**



[Der Durchmesser erhöht sich nicht so sehr, wie die Anzahl der Darstellbaren Knoten gegenüber dem normalen Hypercube]

155. **Welche verschiedenen Bustechnologien kennen Sie?** [Normaler Bus, Mehrfachbus, Hierarischer Bus]

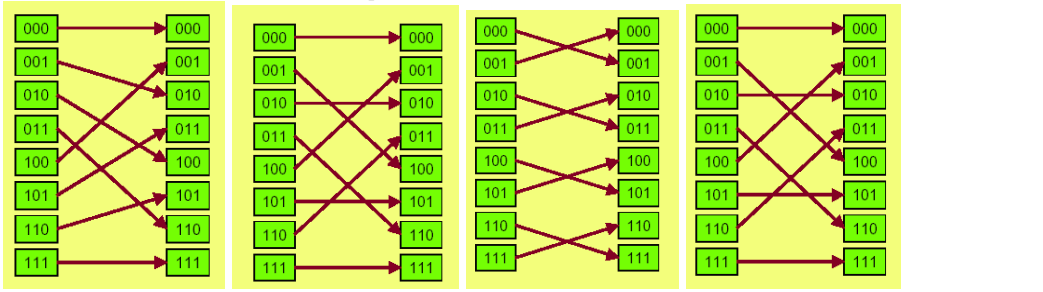
156. **Wie funktioniert ein Crossbar Switch?** [Die Prozessoren werden mit dem Speicher über den Crossbar verbunden, d.h. einzelne Verbindungen gesetzt



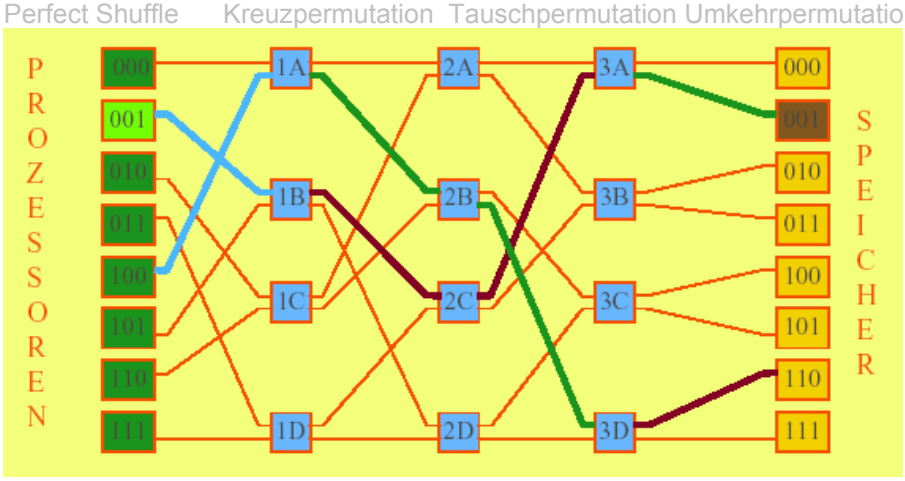
]

157. **Welche verschiedenen dynamischen Verbindungsstrukturen (sog. Permutationsnetzwerke) kennen Sie?** [Perfect Shuffle|Kreuzpermutation|Tauschpermutation|Butterfly|Omega-Netzwerk|Delta-Netzwerl|Butterfly-Netzwerk|Banyan-Netzwerl|Benes-Netzwerk(gespiegeltes Butterflynetzwerk)]

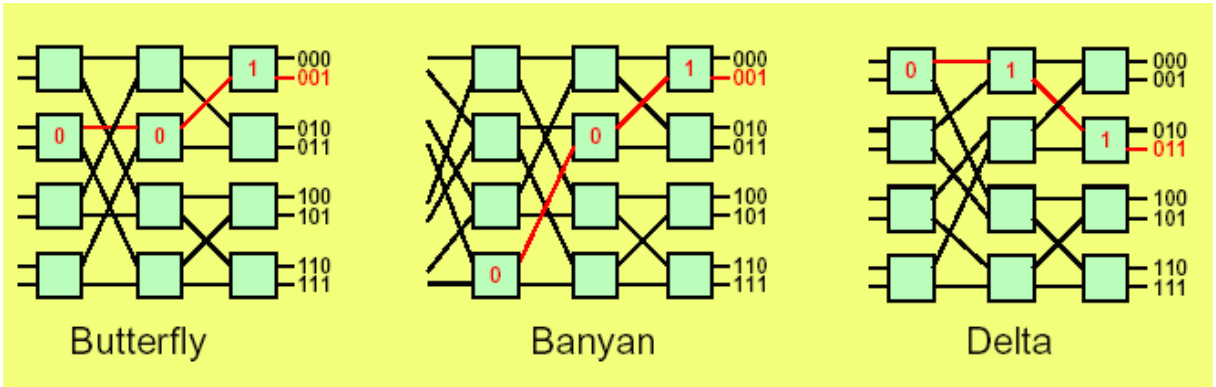
158. **Wie sehen diese Netze aus?** [



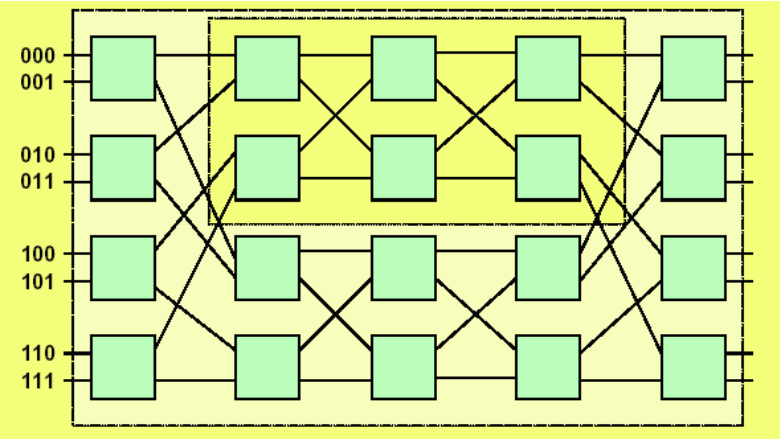
Perfect Shuffle Kreuzpermutation Tauschpermutation Umkehrpermutation (Butterfly)



Omega Netzwerk. Nicht jede Kommunikation ist gleichzeitig möglich, da sich einzelne Kanäle evtl. blockieren



Mischformen➔Zieladresse liefert direkt die Wegeinformation und zwar in der Form, dass eine „0“ bedeutet „oberer Ausgang“ – „1“ bedeutet „unterer Ausgang“



Benes Netzwerk➔Zwei gespiegelte Butterfly Netzwerke➔Jede Permutation kann konfliktfrei geschaltetet werden (Anmerkung: Bei zwei in Serie geschalteten Butterfly Netzen geht das nicht)➔Die Pfadbestimmung ist leider sehr komplex und muss offline durchgeführt werden]

159. **Was versteht man bei diesen Netzen unter „Self-Routing-Property“?** [Die Zieladresse bestimmt den Weg, also zB 0 bedeutet oberer Ausgang (Weg) bzw. 1 unterer Ausgang]
160. **Welche beiden Vermittlungsarten gibt es?** [Paketvermittlung und Leitungsvermittlung]
    a. **Welche beiden Adressierungsarten gibt es zum routen?** [Zielbasiert(Destination Based routing) ➔ wie im Internet | Quellenbasiert (Source Based routing)➔ Die Wege die gewählt werden müssen werden bereits in der Quelle berechnet und ins Paket gepackt]
    b. **Was unterscheidet deterministische Wegewahl und adaptive Wegewahl?** [Deterministisch➔ Wege fest, daher einfache Pfadberechnung, aber keine alternativen bei Blockierung des Pfads
    Adaptive Wegewahl➔Wege dynamisch, daher höherer Hardwareaufwand, aber dafür Alternativwege]
    c. **Was ist ein Phit?** [Eine einzelne Übertragunsgeinheit (→Datagramm beim Internet)]
161. **Was ist der Store-and-forward Modus bei der Datenübertragung?** [Nachricht wird von jedem Zwischenknoten in Empfang genommen, volständig zwischengespeichert und dann weiter übertragen]
162. **Was versteht man bei der Datenübertragung unter „Virtual-Cut-Through“?** [Im Prinzip wie die Virtuelle Verbindung bei Internetkommunikation. Bei blockierten Wegen wird in den einzelnen Knoten zwischengespeichert, bis der Weg frei ist]
163. **Was versteht man unter dem „Wormhole-Routing-Modus“?** [Wie virtual-Cut-Through, aber im Blockierungsfall bleibt der Kopf der Nachricht an der blockierten Stelle im Puffer, der Rest der Nachricht verharrt an der Stelle an der sie gerade sind (wie bei einem Wurm der nicht weiter kommt)]
    a. **Es gibt auch gepuffertes Wormhole Routijng, was ist der unterschied zum normalen?** [Größere Puffer die mehr Paket aufnehmen können ➔ Der Wurm rutscht nach]
164. **Welche Möglichkeiten gibt es die Leistung eines Rechners zu bewerten?** [MIPS|Benchmarks]
    a. **Welche Vor- und Nachteile hat das MIPS Maß?** [Vorteil: einfach verständlich Nachteil: Hängt vom Befehlssatz ab, Keine Berücksichtigung des Speichers und Betriebssystems➔ Schnellere Rechner können niedrigere MIPS Zahl haben]
    b. **Warum sind Benchmarks besser als die reine MIPS Bezeichnung?** [Es werden reale Programme auf den Rechnern ausgeführt, dadurch werden Speicher, I/O, Betriebssystem, Compiler etc. berücksichtigt. Vergleich typischer

Anwendungen➔Spec-Benchmarks, TPC-Benchmarks, DIN-Benchmarks

	DEC Alpha21262	MIPS R10000	IBM PPC750	SUN UltraSparc	PII Klamath	PII Xeon	AMD K6	AMD K6-3D
Taktfrequenz	667	250	400	333	300	400	300	350
SPECint95	44	14.7	17.6	14.2	11.9	16.5	?	?
SPECfp95	66	24.5	12.2	16.9	8.6	13.7	?	?
Transistoren (Mio)	15.2	6.8	6.35	5.4	7.5	7.5	8.8	9.3
Leistungsverbrauch (W)	72	>30	5.7	<30	?	23.3	?	?

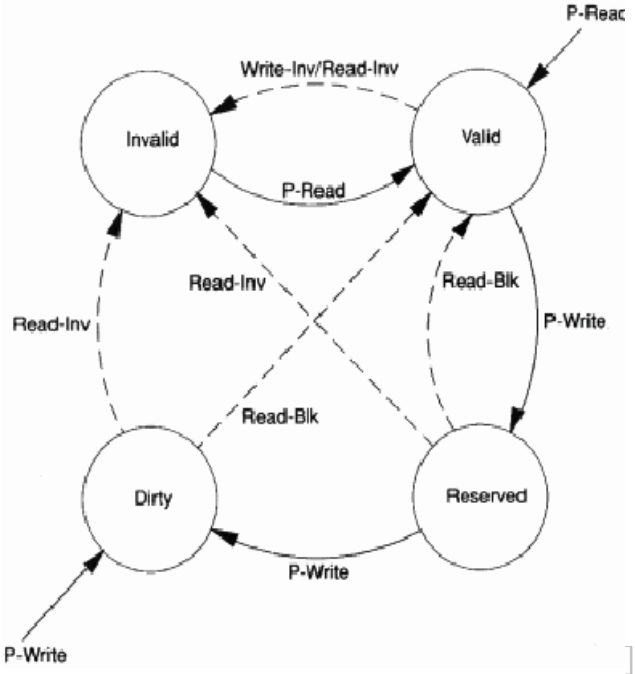
]

165. **Wie misstman nun den Gewinn durch den Einsatz eines Parallelrechners?** [Speed-Up (nach Lee): S(n)=T(1)/T(n) T(1) ist die Ausführungszeit auf einem Singleprozessorsystem in Schritten, T(n) entprechen auf einem Multiprozessorsystem. Anstatt der Ausführungszeit kann man auch die Zahl der auszuführenden Operationen setzen]
    a. **Kann der Speedup auch superlinear sein?** [Hängt von Definition ab bzw. dem verwendeten Algorithmus, ein spezieller paralleler Algorithmus läuft auf einem Seq-Rechner möglicherweise nicht optimal]
    b. **Was kann man mit der Hilfe von Amdahles Gesetz berechnen?** [Speed-Up bei einer bestimmten Anzahl von Prozessoren und Grenzwertig auch den maximalen Speedup in Abhängigkeit vom vorhandenen seriellen Anteil im Algorithmus

$$T(n) = f\,T(1) + (1 - f)\frac{T(1)}{n}$$

    f ist der Anteil des Programms der nicht parallelisiert werden kann (z.B. Filezugriffe auf eine Platte). Bei der Verwendung beliebig vieler Prozessoren, nähert sich der Speedup dem Grenzwert 1/f an]
    c. **Welche weiteren Leistungskennzahlen kennen Sie?** [Effizient, Redundanz, Parallelindex, d.h. Anzahl der möglichen parallelen Operationen pro Zeiteinheit]
166. **Welche beiden Cache-Kohärenzprotokollprinzipien für Multiprozessorsysteme gibt es?** [Directory Based➔Informationen über einen Block befinden sich nur an einer Stelle➔ Nachteil: Kohärenzinformation ist im Prinzip proportional zu der Anzahl der Blöcke im Hauptspeicher |
Snooping: Information über Blöcke befindet sich beim jeweiligen Block im jeweiligen Cache. Jeder Cache-Controller betreibt Snooping am Bus, um zu überwachen, was mit Block passiert➔ Erfordert gemeinsamen Bus➔Snooping ist das übliche Verfahren]
    a. **Welche Snooping Protokolle kennen Sie?** [Write-Broadcast➔ nach dem Ändern wir der geänderte Block an alle anderen Prozessoren geschickt, die ihre Kopie dann ggf. aktualisieren können |
    Write Invalidate: wenn ein Prozessor einen Block verändern will, macht er voprher alle anderen Kopien dieses Blocks ungültig. Beide Protokolle werden verwendet!!
    b. **Wie funktioniert das Write-Once Protokoll?** [➔Cache-Block wird beim Ersetzen in den Speicher zurückgeschrieben, fall er, nachdem er in den Cache geschrieben wurde, verändert wurde(=Zustand Dirty). Es gibt 4 verschieden Zustände: Invalid(Inkonsistente Cachekopie),Valid(Cachekopie ist mit Speicher konsitent), Reserved(Cachekopie mit Speicher kosnistent und einzige Kopie, d.h. Daten wurden

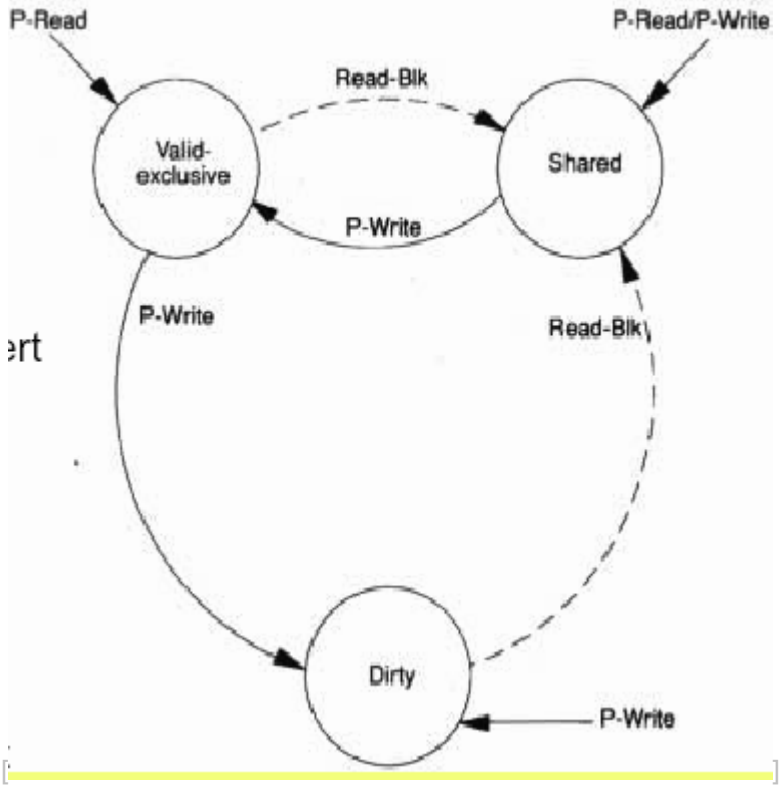
zum ersten mal geschrieben), Dirty: Einzige Kopie und Daten mehr als einmal



verändert]

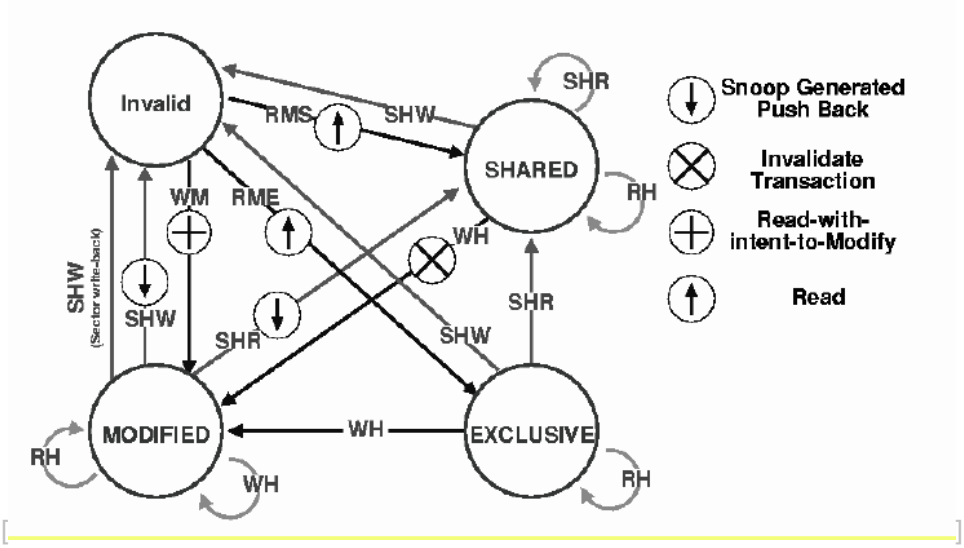
i. **Welche Hit/Miss Situationen gibt es bei „Write Once“?** [Treffer beim Lesen➔ Kann lokal im Cache abgewickelt werden, keine Zustandsübergänge erforderlich
Miss beim Lesen: Keine Kopie im Dirty Zustand➔ Block wird aus Speicher gelesen und erhält den Zustand=Valid | Kopie im Dirty Zustand: ➔Cache mit Dirty Kopie verhindert, das der Speicher eine Kopie schickt und schickt selbst seine Kopie zum anfordernden Cache➔ Speicher wird aktualisiert, und beide Zustände auf Valid gesetzt
Treffer beim Schreiben: Kopie im Dirty oder Reserved Zustand➔ Schreiben kann lokal erfolgen und Zustand wird auf Dirty gesetzt | Kopie im Valid Zustand: Write Invalidate wird ausgelöst➔ Speicherkopie wird aktualisiert und Zustand auf reserved gesetzt
Miss beim Schreiben: Kopie kommt aus Speicher oder aus anderem Cache mit Kopie im Dirty Zustand➔ Wird durch Read invalidate ausgeöst und Speicher wird aktualisiert➔ Neuer Zustand ist DIrty
Ein Block wird ersetzt und in den Speicher zurückgeschrieben, falls er im Dirty Zustand ist. Sonst pasiert nichts]

c. **Wir funktioniert Firefly?**



[ ]

d. **Wie funktioniert das Pentium-MESI Protokoll?**



[ ]

e. **Welche Directory Based-Protokolle kennen sie?** [Bit-Vector-Protokoll|Dynamic Pointer Allocation|Scalable Coherent Interface]

f. **Kennen Sie auch rein Software-basierte Cache Kohärenzverfahren?** [Cacheability Marking|Cache Coherency Enforcement]