KB = 2^{10}, MB=2^{20}

$Takt frequenz = \frac{1}{Zyklusdauer}; Zyklusdauer = \frac{1}{Takt frequenz}$ z.B.2ns = 0.5GHz

	Hz	s	
10^3	k	m	10^{-3}
10^6	M	μ	10^{-6}
10^9	G	n	10^{-9}

Amdahl

$$Dauer_{neu} = Dauer_{alt} \cdot \left(Anteil_{unbenutzt} + \frac{Anteil_{benutzt}}{Beschleunigung_{benutzt}}\right)$$

$$Beschleunigung = \frac{1}{Anteil_{unbenutzt} + \frac{Anteil_{benutzt}}{Beschleunigung_{benutzt}}}$$

Fehlertoleranz

Wahrscheinlichkeit (nicht parallel): A_{sytem} = Produkt aller Einzelwahrscheinlichkeiten

Wahrscheinlichkeit TMR: $A_{sytem} = (3p^2 - 2p^3) \cdot p_{voter}$

Allg: mind. x-1 aus x fehlerfrei: $A_{sytem} = \underbrace{p^x}_{\text{x aus x fehlerfrei}} + \underbrace{\binom{x}{x-1} p^{x-1}(1-p)}_{\text{x-1 aus x fehlerfrei}}$

Quadratische Gleichungen:

$ax^2 + bx + c = 0$

$x_{1/2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Codes:

gerade Parität: gerade Anzahl an 1

Hamming-Abstand h: Anzahl der Bits die sich unterscheiden

Hamming-Abstand bei Code: kleinster Hamming-Abstand der Codewörter

Erkennung von d Bitfehlern: h=d+1; Korrektur von d Bitfehlern: h=2d+1

Erstellen von Hamming-Code, Einzelbitkorrektur:

m= Datenbits, r=Prüfbits

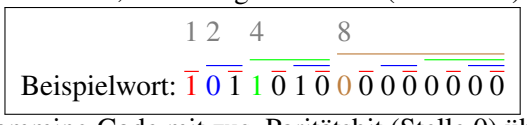
Suche r mit $(m + r + 1) \leq 2^r$

Nummeriere Bits von links mit 1 beginnend durch

Bits mit 2er-Potenznummern (1,2,4,8,...) sind Paritätsbits; aufteilung nach binär (5 ist 1 + 4)

Bsp: Gerade Parität mit $m = 10$

$10 + r + 1 \leq 2^r \Rightarrow r = 4$



SECDED zum Erkennen von 2-Bit Fehlern: Hamming Code mit zus. Paritätsbit (Stelle 0) über das Ganze Wort

Festplatte

Kapazität = $\mathbf{2} \cdot Scheiben \cdot Spuren \cdot Sektoren \cdot Sektorengröße$

$Geschwindigkeit_{\text{außen}} > Geschwindigkeit_{innen}$

$mittlereZugriffszeit = Sektoren \cdot (t_{Spurwechsel} + t_{HalbeUmdrehung})$

$Verschnitt = Sektorgröße - Rest(letzterSektor)$

SSD

Lebensdauer= $\frac{\text{mögliche Schreibzyklen}}{\text{Schreibvorgänge pro Zeit}}$

mögliche Schreibzyklen: verfügbare Zellen * Schreibzyklen pro Zelle (= TBW)

Bei Wear-Leveling:

1. ohne: Lebensdauer = $\frac{\text{Schreibzyklen pro Zelle}}{\text{Schreibvorgänge pro Zeit}}$ (verfügbare Zellen = 1)
2. dynamisch: verfügbare Zellen: Gesamtkapazität - statische Daten
3. statisch: mögliche Schreibzyklen: gesamte SSD

Bus

Steuer-, Adress-, Datenpins, (+Interruptpins)

Bus-Protokoll: Wer legt wann welches Signal an;

Bus-Master: kann Transfer einleiten, Bus-Slave: passiv (Rolle von Kommunikation abhängig)

Bus-Breite: Anzahl der Leitungen

Synchroner Bus: zentraler Takt $\Rightarrow$ Zeiträume aus Graph ablesen

Asynchroner Bus: Buszyklen variabler Länge $\Rightarrow$ Signaländerungen lösen Reaktionen aus

Bus-Arbitrierung: zentral: Bauteil (Arbiter) teilt Buszugriffe zu. (nach Prioritäten)

dezentral: Bauteile prüfen selbst ob sie die höchste angeforderte Priorität haben

Blocktransfer: holen von x Worten ab Adresse

Mikroarchitektur

Minimale Taktzyklusdauer: Summe aller Verzögerungen (z.B. Register,ALU,Shifter ...)

Mic-Architekturen

Befehle=1 Byte, varnum=1 Byte, offset=2 Byte (2er-Komplement) Kürzel: rd (read) wr (write) fetch (Befehl holen)

Register

MAR	Adressregister (Speicherzugriff)
MDR	Datenregister (Speicherzugriff)
PC	Program Counter
MBR	nächster Befehl
SP	Zeiger auf oberstes Stackelement
LV	Lokaler Variablenrahmen (Variablenadressen relativ hierzu)
CPP	Zeiger auf Konstantenbereich
TOS	Oberstes Stapelelement
OPC	alter Program Counter (für Sprungbefehle)
H	Halteregister für Zwischenwerte

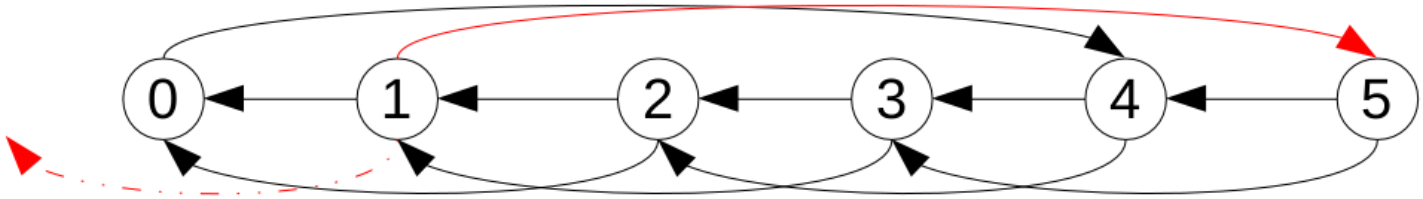
Hex	Mnemonic	Meaning	Mic-1 mit main1	Mic-2
0x00	NOP	Do nothing	2	1
0x10	BIPUSH *byte*	Push byte onto stack	4	2
0x13	LDC_W *index*	Push constant from constant pool onto stack	8	3
0x15	ILOAD *varnum*	Push local variable onto stack	6	3
0x36	ISTORE *varnum*	Pop word from stack and store in local variable	7	5
0x57	POP	Delete word on top of stack	4	3
0x59	DUP	Copy top word on stack and push onto stack	3	2
0x5F	SWAP	Swap the two top words on the stack	7	6
0x60	IADD	Pop two words from stack; push their sum	4	3
0x64	ISUB	Pop two words from stack; push their difference	4	3
0x7E	IAND	Pop two words from stack; push Boolean AND	4	3
0x80	IOR	Pop two words from stack; push Boolean OR	4	3
0x84	IINC *varnum const*	Add a constant to a local variable	7	3
0x99	IFEQ *offset*	Pop word from stack and branch if it is zero	T:11,F:9	T:8,F:6
0x9B	IFLT *offset*	Pop word from stack and branch if it is less than zero	T:11,F:9	T:8,F:6
0x9F	IF_ICMPEQ *offset*	Pop two words from stack; branch if equal	T:13,F:11	T:10,F:8
0xA7	GOTO *offset*	Unconditional branch	7	4
0xAC	IRETURN	Return from method with integer value	9	8
0xB6	INVOKEVIRTUAL *disp*	Invoke a method	23	11
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index		

Mic-1

Jede Mikroinstruktion zeigt auf die folgende im Steuerspeicher, am Ende der Instruktion Verweis auf Main1

Mic-2

Einfädeln von Main1 in die Mikroinstruktionen, vollständiger A-Bus, Instruction Fetch Unit (Hochzählen von PC)
Prefetch: Puffern des Instruktionsstromes in Schieberegister:
Endlicher Automat (Mindestgröße= Wortgröße + längste Instruktion/Operand -1) z.B. bei IJVM 4+2-1 = 5



Mic-3

3-stufige Pipeline: Lesen, ALU, Schreiben (RAW-Abhängigkeiten der Mikroinstruktionen beachten)

Mic-4

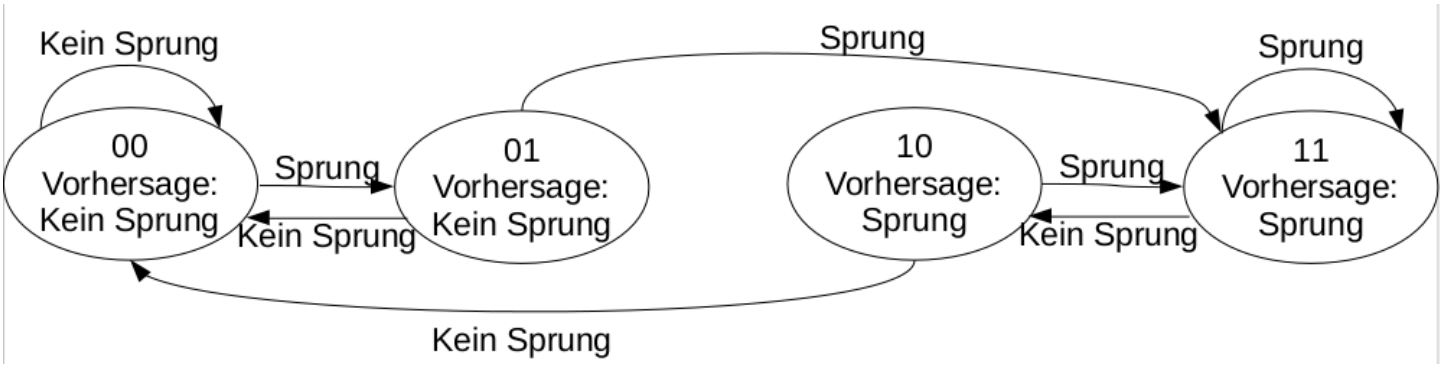
7-stufige Pipeline

1. Instruction Fetch Unit
2. Dekodiereinheit: Zerlegt Instruktionen in Opcode/Operanden
3. Queueing: Füllt Mikroinstruktionswarteschlange aus ROM-Tabelle(aufeinanderfolgende Mikroinstruktionen)
4. Lesen
5. ALU
6. Schreiben
7. Speicherzugriff

Sprungvorhersage

Behandlung falscher Sprünge: Schreiben nur auf Schattenregistern (=> rückgangigmachen möglich)

- Einfache Methode: sprünge Rückwärts wahrnehmen, Vorwärts nicht
- Statische Sprungvorhersage: Compiler gibt empfehlung
- Dynamische Methode: History-Tabelle (analog Cache): valid, Tag, Entscheidungs-Bits
  => Entscheidungs-Bits anhand Endlichem Automaten, z.B. (Wechsel nach 2 Fehlvorhersagen)



Ausführung außer der Reihe

Abhängigkeiten: RAW (Read-After-Write) WAW, WAR
WAW, WAR => Lösung: Schattenregister
Beachten: Änderung der Reihenfolge möglich?
Bsp: (2 Ausführungseinheiten, 1 Zyklus pro Instruktion:)

1. MUL R1, R2, R1
2. ADD R4, R5, R6
3. SUB R3, R1, R7
4. MUL R8 R3, R3
5. DIV R11, R4, R4
6. ADD R15, R14, R13

⇒ Abhängigkeiten: RAW: 3-1, 4-3, 5-2

⇒

Keine Änderung der Reihenfolge:
1: 1 2
2: 3
3: 4 5
4: 6
Änderung der Reihenfolge:
1: 1 2
2: 3 5
3: 4 6

Scheduling: (2 Ausführungseinheiten und dekodieren von 2 Instruktionen pro Zyklus)

					Read Registers					Write Registers					
Cy	#	Decoded	Issued	Retired	0	1	2	3	4	0	1	2	3	4	
1	1	R3 = R0 * R1	1		1	1							1		
	2	R4 = R0 + R2	2		2	1	1						1	1	
2	3	R1= R4 + R1	-		2	1	1						1	1	<= RAW
	4	R5 = R6 + R7	4		2	1	1						1	1	
3		*bei Ausführung außer der Reihe möglich*		1	1		1							1	
				2											
4			3			1			1		1				
						1			1		1				
5						1			1		1				
						1			1		1				
6				3											

ISA

Instruktionsformate:

Opcode +[Operand/en], evtl. gleiche Länge

Addressiermodi

- Umittelbar: Konstanter Wert (z.B. 4)
- Direkt: Speicherstelle (z.B. 0x4568AF)
- Indirekt: Adresse der Speicheradresse (Zeiger)
- Register: Registerinhalt
- Indirekte Register: Adresse steht im Register (z.B: Stackpointer)
- Indiziert: Offset von fester Adresse (z.B. Arrays)
- Basis-Indiziert: Offset von Registerinhalt
- Stapel: vgl. Postfix-Notation

Für Sprünge:

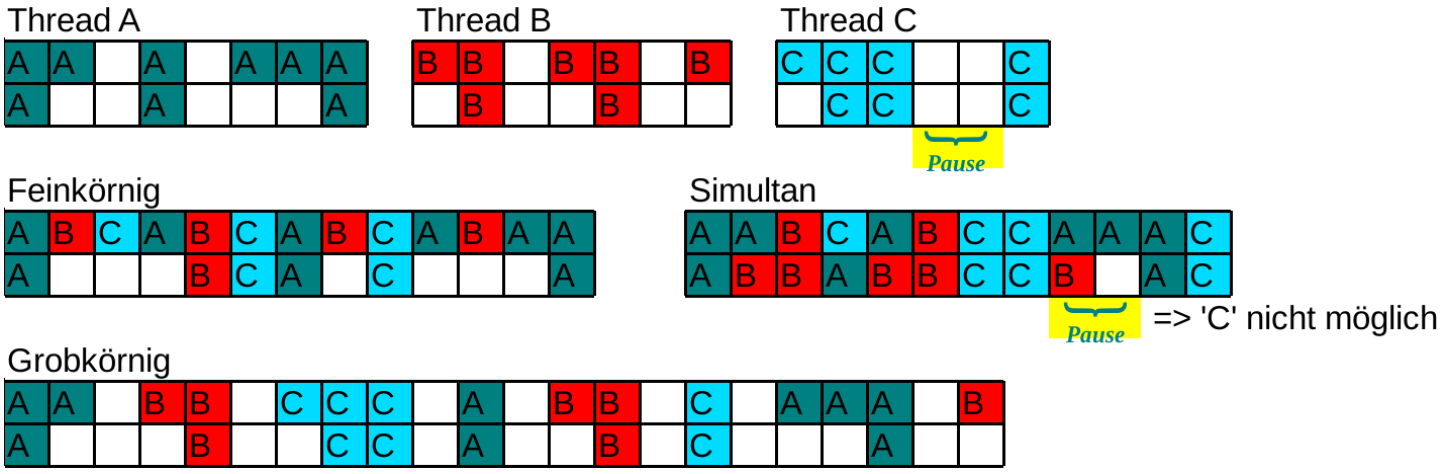
- Direkt
- Indirekte Register
- Indiziert
- PC-relative: Offset zum Program Counter

Postfix-Notation:

Operator hinter Operanden
z.B. A+BxC => ABCx+
(A+B)/(C-D) => AB+CD-/

Parallelität

On-Chip-Multithreading: Mehrere Threads pro CPU (gegen Pipelinestalling)
Beachten: Lücken innerhalb eines Threads (vgl Pause in Abb.)
Feinkörnig: Abwechselndes Ausführen (Jeden Takt ein anderer Thread); nur 1 Thread pro Takt
Grobkörnig: ausführung des Threads bis Leertakt eintritt (evtl. Wechselverzögerung beachten); nur 1 Thread pro Takt
Simultan: vgl. Grobkörnig, Threadwechsel innerhalb eines Taktes möglich



Cache

Split Cache: Trennung Instruktions-/Daten-Cache (vs Unified-Cache)
Inclusive Cache: Inhalt auch in niederer Stufe enthalten (L1 $\subset$ L2)
Victim Cache: nimmt Zeilen auf, wenn aus höherer Stufe entfernt;
Exclusive Cache: löscht Zeilen, wenn in höhere Stufe gegeben

Cache-Schreiben (Cache-Hit):
Write-Through: aktualisiere HS sofort bei schreiben im Cache
Write-Back: aktualisiere HS erst bei entfernen der Cache-Zeile

Cache-Schreiben (Cache-Miss):
Write-Allocation: Hole in Cache und schreibe dann
Write-Around: Schreibe nur in der unteren Ebene

Cache-Zugriffszeiten

h:= Cache-Trefferquote:

Wenn L2 $\underline{30\%}$ der Misses aus L1: $h_{L2} = \underline{0,3} \cdot (1 - h_{L1})$
Wenn L3 $\underline{40\%}$ der Misses aus L2: $h_{L3} = \underline{0,4} \cdot (1 - h_{L1} - h_{L2})$

Parallele Zugriffszeit= $h_{L1} \cdot t_{L1} + h_{L2} \cdot t_{L2}$
Sequenzielle Zugriffszeit= $h_{L1} \cdot t_{L1} + h_{L2} \cdot (t_{L1} + t_{L2})$

Direkt abgebildet

Eine Zeile = Ein Eintrag mit: Valid-Bit(s), Tag, Daten
Zerlegen der Adressen in | Line | Tag | Word/Byte |

|Word/Byte|	$2^5 \frac{Byte}{Zeile} => 5Bit$
|Line|	$Cache : 2^{13}Byte, 2^5 \frac{Byte}{Zeile} => 2^{13-5} = 8Bit$
|Tag|	32Bit Adressen, 8 Bit Line, 5 Bit Word/Byte => (32-8-5) = 19 Bit

1.Byte/Wort: Word/Byte = $(0 \ldots 0)_2$
letztes Byte: Word/Byte = $(1 \ldots 1)_2$
letztes Wort: Word/Byte = $(1 \ldots 100)_2$
Zugriff:

1. Suche Zeile Nummer #Line#
2. prüfe valid ?
3. prüfe Tag = Tag-Eintrag

Teilassoziativer n-Wege Cache

Bis auf Line wie Direkt Abgebildet;
Eine Menge($\tilde{\text{L}}$ine) enthält mehrere Cache-Zeilen (jede mit Tag-Eintrag und Valid-Bit)
Berechnung: |Line| aus Cachegröße, zeilenlänge und Anzahl Wege
z.B. $\frac{2^{13}Byte\text{Kapazität}}{2^5 \frac{Byte}{Zeile} \cdot 4\text{Wege}} = 2^6 => 6Bit$

Kohärenzprotokolle

Lösen das Problem Verschiedener Versionen der selben Cache-Zeile **Zustände:**
Invalid: Zeile **nicht** im Cache
Shared: Zeile in einem oder mehreren Caches (bis MESI im HS aktuell)
Modified: im lokalen Cache aktuelle Version, alle anderen invalid (auch HS)
Exclusive: im lokalen Cache und HS, sonst keine Kopien
Owned: im lokalen Cache **nicht** im HS, (shared) Kopien in anderen Caches
Forward: vgl. Shared, Letzter Lesezugriff ist Forward (spart HS-lesen)
Protokolle
SI: bei Schreiben wird invalidiert (alle Caches)
MSI: bei Schreiben wird Modified, die anderen Caches invalidieren
MESI: wenn einzige kopie (beim lesen in den Cache), dann Exclusive sonst Shared
MOESI: wenn M und read durch anderen Cache: O
MESIF: der letzte lesende Zugriff (shared) ist F