

| Pipelining                                                                                                          |                                                                 |
|---------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| <b>Instruction Fetch (IF)</b>                                                                                       | nimmt Befehle vom HS auf                                        |
| <b>Instruction Decode (ID)</b>                                                                                      | decodiert Befehle & Register; schaut in Register & lädt Inhalte |
| <b>Execute (EX)</b>                                                                                                 | führt Operation aus                                             |
| <b>Memory Access (MA)</b>                                                                                           | macht L/S Befehle im HS                                         |
| <b>Write Back (WB)</b>                                                                                              | schreibt in Register                                            |
| <b>Vorteil:</b> Parallelisierung von Operationen                                                                    |                                                                 |
| <b>Nachteil:</b> bei Pipelineflush müssen alle Befehle verworfen und neu geladen werden; ohne Cache immer Cachemiss |                                                                 |

| Konflikte                |                                                   |
|--------------------------|---------------------------------------------------|
| <b>Write after Write</b> | doppelte Überschreibung                           |
| <b>Write after Read</b>  | Wert, der noch gebraucht wird, wird überschrieben |
| <b>Read after Write</b>  | falscher Wert wird gelesen                        |

| Stalling                            |                                                                                      |
|-------------------------------------|--------------------------------------------------------------------------------------|
| <b>Pipeline Stalling</b>            | lässt Operationen einen Takt warten                                                  |
| <b>Result Forwarding</b>            | Wert direkt weitergeben; <i>kein Zeitverlust, geht aber nur wenn Wert erreichbar</i> |
| <b>Speculative Execution:</b> raten |                                                                                      |

| Befehlstypen  |            |
|---------------|------------|
| Abacus / MIPS |            |
| R / R         | 3 Register |
| I / I         | 2 Register |
| S / -         | 1 Register |
| J / J         | 0 Register |

| Stalls  |         |        |
|---------|---------|--------|
|         | ohne FW | mit FW |
| arithm. | 2       | 0      |
| L/S     | 2       | 1      |
| Branch  | 3       | 1      |
| Jump    | 3       | 0      |

| Nakata                                                                  |  |
|-------------------------------------------------------------------------|--|
| * findet min. Anzahl an Registern, die für ein Programm nötig sind      |  |
| * <b>Knoten == Variable:</b> set RegNum = 1                             |  |
| * <b>Knoten unäre Operation</b> und Kind Label r hat: set RegNum = r    |  |
| * <b>Knoten binäre Operation:</b> *                                     |  |
| $l = r$ : set RegNum = l + 1 = r + 1                                    |  |
| * $l \neq r$ : set RegNum = max{l, r}                                   |  |
| * funktioniert nicht bei geteilten Registern, z.B.: (c * x + b) * x + a |  |

| Parsertabelle                                                                                                  |  |
|----------------------------------------------------------------------------------------------------------------|--|
| <b>a)</b> ist das T in unserer FIRST-Menge drin? Dann suche Produktion, welche das T in die FIRST-Menge bringt |  |
| <b>b)</b> ist e in FIRST, dann schaue, ob unser T auch in der FOLLOW-Menge drin ist                            |  |
| Falls ja, nehme e auf                                                                                          |  |

| Byteorientierter Cache Beispiel |                       |          |  |
|---------------------------------|-----------------------|----------|--|
|                                 | tag   adr             | valid    |  |
| 15 sti \$1, \$0, 0              | 1 in adr[0] -> 00 00  | 0 -> 1   |  |
| 16 sti \$2, \$0, 1              | 2 in adr[1] -> 00 01  | 0 -> 1   |  |
| 17 ldi \$2, \$0, 1              | Mem[1] laden -> 00 01 | 1 -> hit |  |
| 18 sti \$3, \$0, 2              |                       |          |  |
| 19 sti \$4, \$0, 3              |                       |          |  |
| 20 ldi \$2, \$0, 1              |                       |          |  |
| 21 sti \$5, \$0, 7              |                       |          |  |

| Byteorientierter Cache Beispiel (cont) |                      |                           |                 |
|----------------------------------------|----------------------|---------------------------|-----------------|
| 22 sti \$6, \$0, 8                     |                      |                           |                 |
| 23 sti \$7, \$0, 9                     | 7 in adr[9] -> 10 01 | valid = 1 && tag[9] != 00 | aus 16 -> miss  |
| 24 ldi Mem[1] \$2, \$0, 1              |                      |                           | Cachemiss laden |

| Bitorientierter Cache                                  |                                |
|--------------------------------------------------------|--------------------------------|
| Datewidth                                              | 16 entspricht 2 Byte Wortgröße |
| MemSize                                                | 32 HS Größe                    |
| Cachesize                                              | 8 entspricht 4 Wörter          |
| Blocksize                                              | 2 entspricht 1 Wort            |
| SetAssoc                                               | 1 entspricht 1 Block pro Satz  |
| <b>direct mapping:</b> jedes Wort genau eine Zuteilung |                                |

| CPI          |     |         |                                     |
|--------------|-----|---------|-------------------------------------|
| Befehlstyp   | %   | mit RAW | CPI                                 |
| Sprungbefehl | 20% | 20%     | 1 bei jump; 2 bei branch            |
| Arithmetik   | 60% | 60%     | 1                                   |
| Ladebefehl   | 20% | 0.5*20% | 1 wenn Wert darauf nicht gelesen, 2 |
| MAC          |     | 0.5*20% | -                                   |

50% der Ladebefehle erzeugen einen RAW Konflikt

$$CPI = CPI_A * 20 + CPI_I * 60 + CPI_L * (20 - 10) / 100 - (20 - 10)$$

| Lokalitäten                                                                                                       |  |
|-------------------------------------------------------------------------------------------------------------------|--|
| <b>räumlich:</b> Adressen, die nah an dem genutzten sind, werden wahrscheinlich wieder genutzt -> <i>Blöcke</i>   |  |
| <b>zeitlich:</b> eine Adresse, auf die zugegriffen wird, wird wahrscheinlich in nächster Zeit wieder angesprochen |  |



By **Everdeen**

[cheatography.com/everdeen/](https://cheatography.com/everdeen/)

---

Published 28th March, 2018.

Last updated 28th March, 2018.

Page 1 of 3.

---

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish

Yours!

<https://apollopad.com>

### Zurückschreibmethoden

**write through:** schreibt immer durch

**write back:** nutzt dirtybit

schreibt zurück, wenn etwas neues eingelagert wird vom HS (oder geladen)

**least recently used:** für set-oriented:

schreibt in Zeile, die am längsten nicht genutzt wurde

### Graph Coloring

**Spilling:** nutzen den HS

1. Berechnen Read / Write-Menge & Zeilen / Blöcke

2. MayLive berechnen

3. MayUsed bestimmen

4. Zeilenweise den Schnitt nehmen

5. alle in der selben Menge haben

Beziehung zueinander

6. Assembler-Code produzieren

\* werden x Farben benötigt, dann ist

Codierung auch mit x Registern möglich

### Cacheadressierung

|           | Byte                                 | Block             | N <sub>k</sub> -set                |
|-----------|--------------------------------------|-------------------|------------------------------------|
| #index    | $\log_2(\text{Cachegr.})$            | $s_k$             | $s_k - n_k$                        |
| #tag      | $\log_2(\text{HSGR}) - \text{\#adr}$ | $a_1 - (s_k + b)$ | $a_1 - (s_k - n_k + b)$            |
| #offset   | -                                    | b                 | b                                  |
| #gl. tags | #Zeilen im Cache                     | size <sub>k</sub> | size <sub>k</sub> / N <sub>k</sub> |
| Sätze     | -                                    | size <sub>k</sub> | size <sub>k</sub> / N <sub>k</sub> |

size<sub>k</sub> = #Blöcke im Cache

s<sub>k</sub>: size<sub>k</sub> = 2<sup>s<sub>k</sub></sup> Blöcke; B = Blockgröße in Byte;

b: B = 2<sup>b</sup>; a<sub>1</sub> = Bits für Adressraum

n<sub>k</sub>: N<sub>k</sub> = 2<sup>n<sub>k</sub></sup>; N<sub>k</sub> = #Blöcke im Satz

Adressberechnung

$\text{adr}(x) = x \bmod (\text{Cachelines}); \text{tag}(x) = x \text{ div } (\text{CL});$

$\text{adr}(x) = (x \text{ div } 2^b) \text{ div } 2^{s_k}; \text{tag}(x) = (x \text{ div } 2^b) \bmod 2^{s_k}$

$\text{adr}(x) = (x \text{ div } 2^b) \text{ div } 2^{s_k - n_k}$

$\text{tag}(x) = (x \text{ div } 2^b) \bmod 2^{s_k - n_k}$

### Blockorientierter Cache

\* direct mapping

\* Laden & Speichern gesamter Blöcke

+ nutzen aus, dass "nahe" Werte abhängig voneinander sind -> kein unnötiges Laden

- blöd, wenn diese unabhängig sind

### Blockorientierter Cache Beispiel

15 sti \$1, \$0, 0

Mem[0] -> 00|0|0 = tag | in welchem Block | in welcher Zeile im Block

21 sti \$5, \$0, 7

Mem[7] -> 01|1|1

Cachemiss 00!=01

-> laden den Block on den HS zurück, um neuen Wert speichern zu können

### Satzassoziativer Cache

\* kein direct mapping

\* darf sich aussuchen, in welchen Block

+ reinspeichern angenehmer, da freie Platzwahl

- suchen aufwändiger, da wir alle Blöcke im Set durchgehen müssen

\* 1-way -> blockorientiert, da nur ein Block und somit keine Platzwahl

### Top-Down-Parsing

| Stack     | input     | action           |
|-----------|-----------|------------------|
| \$S       | bbabbbb\$ | expand S -> bSbb |
| \$bbSb    | bbabbbb\$ | match            |
| \$bbS     | babbbb\$  | expand S -> bSbb |
| \$bbbbbSb | babbbb\$  | match            |
| \$bbbbbS  | abbbb\$   | expand S -> A    |
| \$bbbbbA  | abbbb\$   | expand A -> aB   |
| \$bbbbbBa | abbbb\$   | match            |
| \$bbbbbB  | bbbb\$    | expand B ->      |
| \$bbbbb   | bbbb\$    | match            |
| \$bbbb    | bbb\$     | match            |
| \$bb      | bb\$      | match            |
| \$b       | b\$       | match            |

### Top-Down-Parsing (cont)

\$ \$ match

S -> bSbb | A

A -> aB

B -> A |

bbabbbb in Sprache?

### First & Follow Beispiel

|   | FIRST | FOLLOW                       |
|---|-------|------------------------------|
| S | b, a  | \$, FIRST(b)\e aus FOL(S): b |
| A | a     | FOL(S) aus FOL(A): \$, b     |
| B | a,    | FOL(A) aus FOL(B): \$, b     |
| a | a     | -                            |
| b | b     | -                            |

### Caches

\* Cache wird bei L/S Befehlen angesprochen

\* alles in binär

**dirtybit:** Cache konsistent zum HS?

Ja: 0; Nein: 1

**validbit:** ist ein Datum in der Adresse?

Ja: 1; Nein: 0

**Blockorientiert:** es werden nur Teile im Block überschrieben

**Set-orientiert:** freie Auswahl in Adresse

**Cache-hit:** valid = 1 & tag(i) == tag(i')

**miss:** valid = 0 || valid = 1 & tag(i) != tag(i')

**Trefferquote:** #hit/#miss

**Average Acces Time:** Hitrate\*time(hit) + (1-Hitrate\*time(miss))

**Miss-Penalty:** extra delay durch HS-Zugriff

### First & Follow

**FOLLOW:**

\$ aus FOL(S)

A->aBb: FIRST(b)/e aus FOL(B)

A->aB: FOL(A) aus FOL(B)

A->aBb, e aus FIRST(b): FOL(A) aus FOL(B)



By **Everdeen**

[cheatography.com/everdeen/](https://cheatography.com/everdeen/)

---

Published 28th March, 2018.

Last updated 28th March, 2018.

Page 2 of 3.

---

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish

Yours!

<https://apollopad.com>

### Data Flow Analyse

**Basic Block:** \* Zeile nach einem if bildet neuen Block  
 \* if goto a bildet neuen Block an Zeile a  
 -> *erste Zeile im neuen Block*  
 -> \*nur bei goto keine Beziehung zum nachfolgenden Block  
**Read-Menge:** nehmen von allen Reads die raus, die wir im eigenen Block vorher schreiben  
**Write-Menge:** alle geschriebenen Variablen

### Variablen

\* auf Klammerung achten  
 \* am Besten Formeln aufschreiben  
**May Live:** \* backwards analysis  
 $Live(I) = Read(I) \cup \{l \mid l \rightarrow I \wedge (Live(l) \setminus Write(l)) \text{ folgender Block} \}$   
**Must Live:** \* backwards  
 $MLive(I) = Read(I) \cup \{l \mid l \rightarrow I \wedge (MLive(l) \setminus Write(l)) \}$   
**MayUsed:** \* forwards  
 $U(I) = Read(I) \cup \{l \mid l \rightarrow I \wedge (U(l) \cup Write(l)) \}$   
**MustUsed:** \* forwards  
 $MustU(I) = Read(I) \cup \{l \mid l \rightarrow I \wedge (MustU(l) \cup Write(l)) \}$   
**Busy:** \* backwards  
 $B(I) = Read(I) \cup \{l \mid l \rightarrow I \wedge (B(l) \setminus Write(l)) \}$

### Pipeline

**CISC** register memory architecture  
*machinewords different length* \*  
*many complex* \* irregular instructions

### Pipeline (cont)

**RISC** *register register architecture* as few as possible & as regular as possible instructions \* machine words same length

shared subtrees in syntaxtrees: DAG  
 -> *optimaler code für DAGs ist NP-vollständig*

**MSB (Most Significant Bit):** linkstes Bit (höchste Potenz)

**LSB (Least):** rechts

**vollasoziiert:** wenn Cache 1 Satz (freie Speicherwahl)

n-fach assoziativ:  $n = N_k$

**Bootstrapping:** Aneinanderschalten von Compilern

-> kein extra Compiler

**L/S-Architektur:** Befehlssatz Daten-Speicherzugriffe nur mit L/S Befehlen

-> RISC

-> CISC hat auch ALU (arithmetic & logic unit)

### Virtueller Speicher

\* **TLB:** Translation-Lookaside Buffer

\* **Pagetable** ist im HS



By Everdeen

[cheatography.com/everdeen/](https://cheatography.com/everdeen/)

Published 28th March, 2018.

Last updated 28th March, 2018.

Page 3 of 3.

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish Yours!

<https://apollopapad.com>