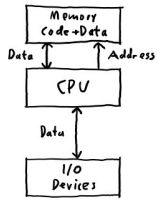
Rechnerarchitekturen

Von-Neumann

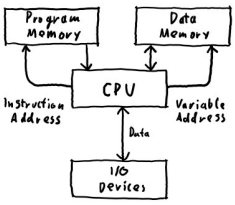


Pros: Flexibilität
Cons: Speed

Example:
General Purpose Computer

1. FETCH INSTRUCTIONS
2. DECODE
3. FETCH OPERANDS
4. EXECUTE
5. WRITE BACK

Harvard



Pros: Speed

Cons: Flexibility

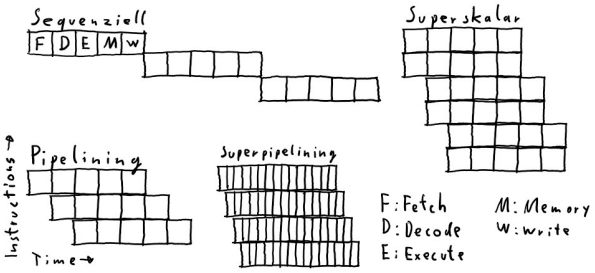
Example:
Embedded Systems

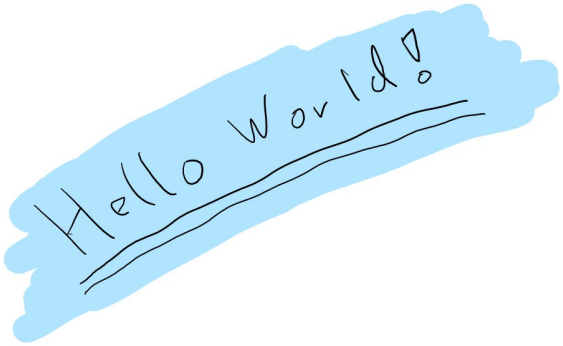
RISC

+ kurze Ausführungszeiten
+ effizienteres Pipelining
+ einfachere / kleinere Befehlssätze
+ simple Adressmodi

CISC

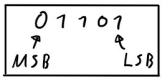
+ kleinere Programme als RISC
+ Konstrukte höherer Programmiersprachen teilweise direkt unterstützt





Speicher

Big Endian: MSB in kleinster Adresse ← Byte
Little Endian: LSB in kleinster Adresse



vector<> ist ein dynamisches Array, list<> eine verkettete Liste.
Vorteil vector<>: kontinuierlicher Speicher, indizierter Zugriff.
Nachteil vector<>: Löschen / Einfügen ist aufwendig.
Vorteil list<>: Einfügen / Löschen an beliebiger Stelle effizient möglich.
Nachteil list<>: Größerer Speicherbedarf für zusätzliche Zeiger.

Cache

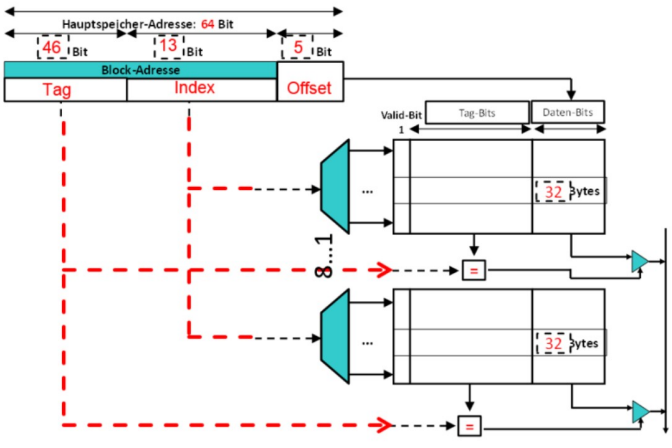
LRU (least recently used)
LFU (last frequently used)
FIFO (first in first out)

n-assoziativer Cache
Blöcke = Cachegröße / Blockgröße
Sets = (Cachegröße / Blockgröße) / n

Offset = ld (Bytes pro Block) → Bit
Index = ld (Datenmenge in Byte / Bytes per Block / Sets → Bit
Tag = Adressbreite − Offset − Index → Bit

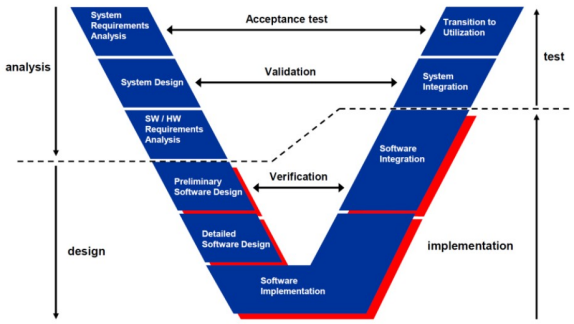
Speichergröße = Datenspeicher + Speicher für (Tags + Valid Bits)

Voll-assoziative Caches sind n-Wege Caches mit $n = m$ (m = Anzahl an Cachezeilen) Cache-Sets. **Nachteil** des voll-assoziativen ist, dass $m - 1$ mehr Komparatoren als bei direkt-abbildenden Caches benötigt werden und damit mehr Fläche verbraucht. Sie haben aber den **Vorteil** der höheren Flexibilität beim Abbilden von Daten aus dem Hauptspeicher in den Cache, da jedes Byte in m verschiedenen Cache Sets gemappt werden kann (höhere Cache Hit-Rate).

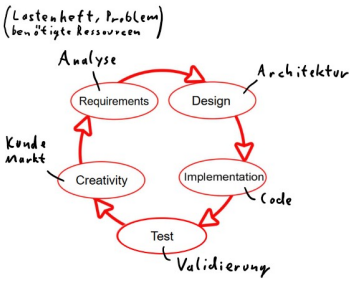


Offset: Adressiert ein Datenwort innerhalb eines Blockes/Zeile.
Index: Adressiert einen Block/Zeile innerhalb des Caches.
Tag: Dient als Referenz im Cache, um die Gültigkeit der aktuellen Hauptspeicheradresse festzustellen.
Valid-Bit: Gibt an, ob im betreffenden Block die Daten gültig sind, d.h. ob der entsprechende Block in Benutzung ist, oder ob der Block frei ist und überschrieben werden kann.

Softwave Engineering



Akzeptanztest: Fertiges Produkt gegen Anforderungen validieren
Verifikation: Prüfen ob Produkt richtig entwickelt wurde
Validierung: Prüfen ob das richtige Produkt entwickelt wurde

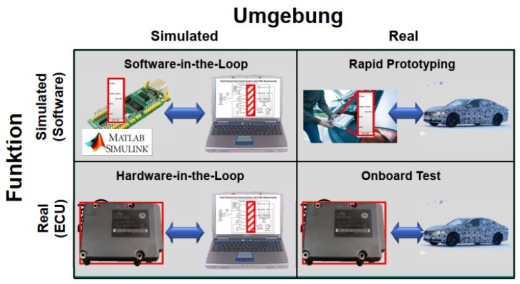


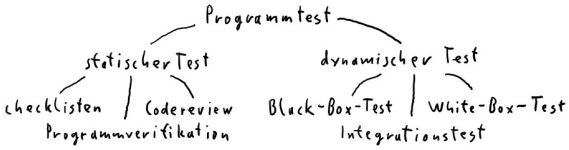
Merkmale guter Software

- Zuverlässigkeit
- Wartungsfreundlichkeit
- Effizienz
- Benutzerfreundlichkeit
- Nützlichkeit

Realtime-Systeme

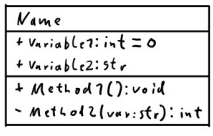
hard-, soft- & non-Realtime





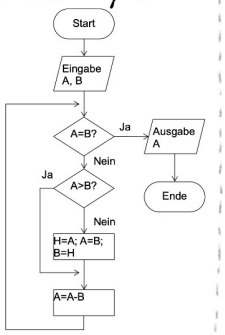
Programmiersprachen

Klassendiagramm

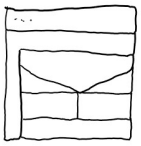


- Öffentlich (+)
- Privat (-)
- Geschützt (#)
- Paket (~)
- Abgeleitet (/)
- Statisch (unterstrichen)

Flussdiagramm



NS-Diagramm



imperativ: Assembler, FORTAN, ALGOL
prozedural: Pascal, C
objektorientiert: C++, Java, C#

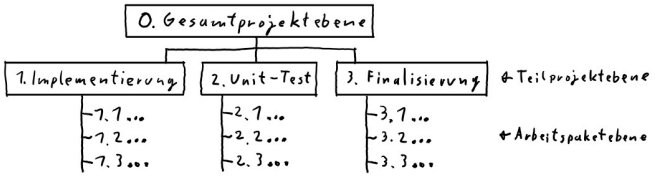
```
bool HashTable::remove(const std::string& name) {

    uint8_t element_hash = hash(str); // Siehe oben
    auto iter = elements.begin(); // Siehe oben
    std::advance(iter, element_hash); // iter auf den h-ten Index setzen
    Entry* entryPtr = *iter;
    // Schauen, ob Element da ist und der Key übereinstimmt
    if (entryPtr != NULL && entryPtr->key == str) {
        delete entryPtr; // Entry vom Heap löschen
        *iter = NULL; // Pointer auf Entry "löschen"
        return true;
    }
    return false;
}

void HashTable::add(const std::string& name, const std::string&
    data) {

    Entry* element = new Entry();
    element->key = name;
    element->value = data;
    uint8_t element_hash = hash(name); // Siehe oben
    auto iter = elements.begin(); // Siehe oben
    std::advance(iter, element_hash); // iter auf den h-ten Index setzen
    *iter = element;
}

std::string HashTable::get(const std::string& name) {

    uint8_t element_hash = hash(str); // max. 256, da ein uint8_t, gibt den
    Indexelements[h]
    //geht nicht, da eine std::list statt einem std::vector/nem array verwendet wird
    auto iter = elements.begin();
    std::advance(iter, element_hash); // iter auf den h-ten Index setzen
    Entry* entryPtr = *iter;
    if (entryPtr == NULL) {
        return "";
    } else {
        return entryPtr->value; // Vermutlich wollen sie hier value, aber kp
    }
}
```

Projektmanagement

Lastenheft (Auftraggeber)
Pflichtenheft (Auftragnehmer)

Rahmenheft (Ideen/Konzepte)
Projektstrukturplan
Projektablaufsplan (Kritischer Pfad/zeitlicher Ablauf)

Projektstrukturplan (PSP)



Datenstrukturen

Array, Liste, Stack, Queue,
Graph, Baum, Heap, Hashtable

* Made on Earth by humans *