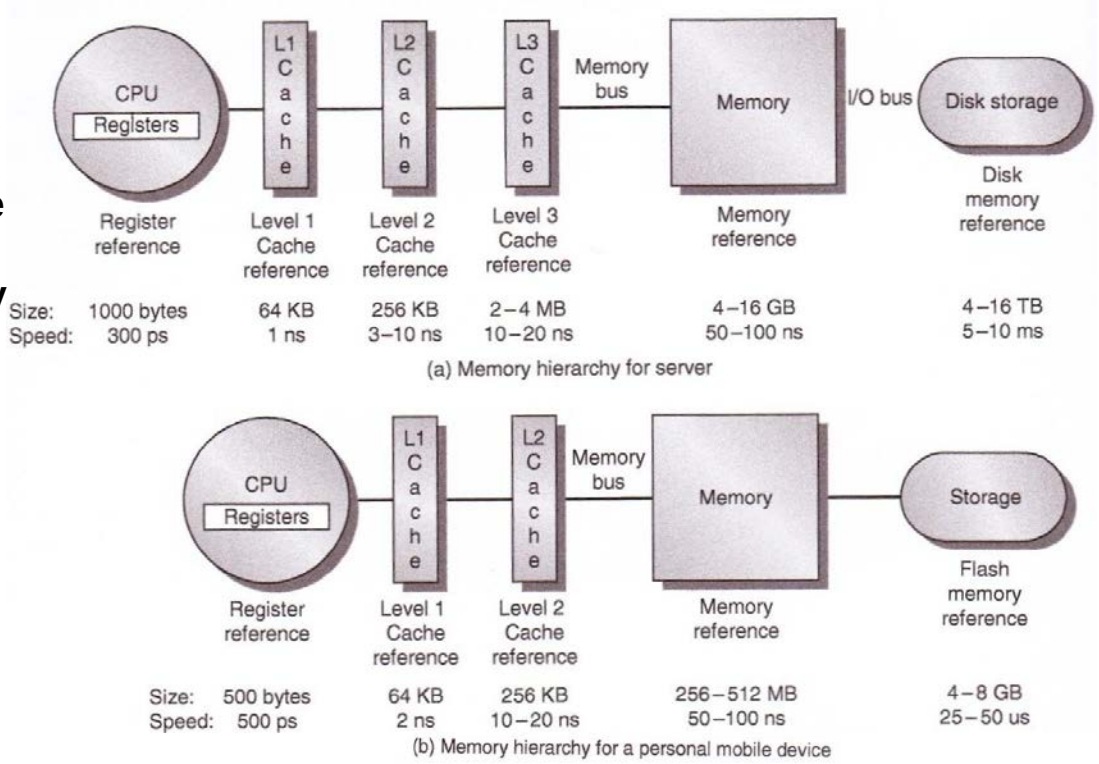
Hierarchie paměťového systému

Břetislav Bakala

Hierarchie pamětí počítače

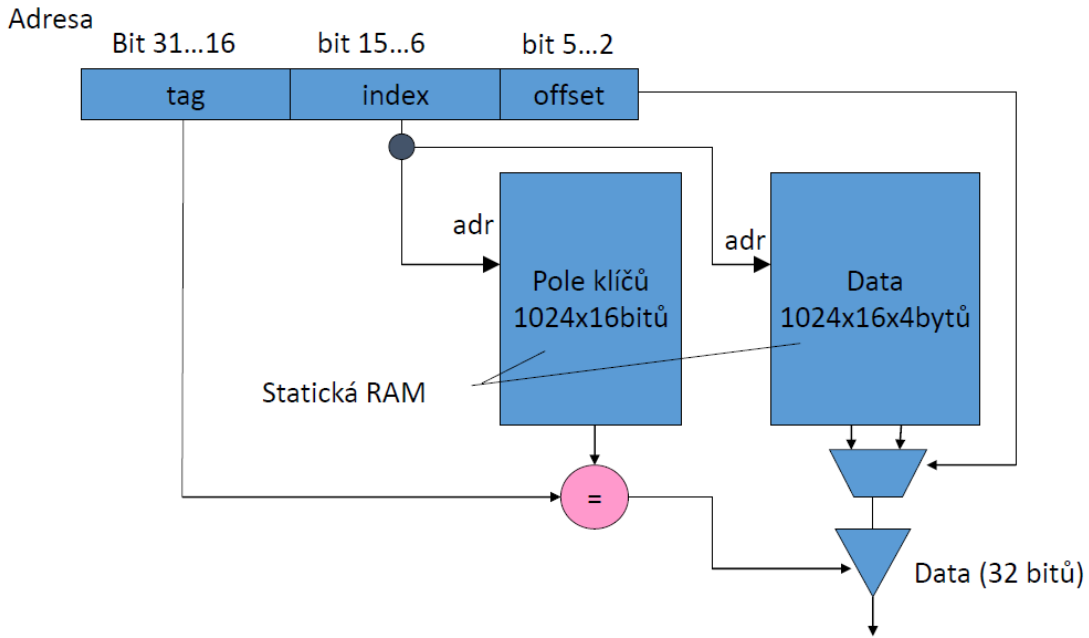
- Požadavek – **neomezené** množství **rychlé** paměti
- **Ekonomické** řešení – **hierarchie** paměťového systému
- Hierarchie využívá výhod **lokality** (temporal, spatial locality)
- je postavena na **kompromisu výkonnosti** paměťových technologií k ceně
- **Nižší úrovně** (blíž k procesoru) jsou **rychlejší a menší** (dražší)
- **Inclusion property** – princip zahrnutí dat paměti nižší úrovně do vyšší úrovně



Přímo adresovaná cache

Základy pamětí cache jsou podrobně probrány v kurzu Architektura počítačů I.

Pouze jeden komparátor
Vícebitový adresový dekodér



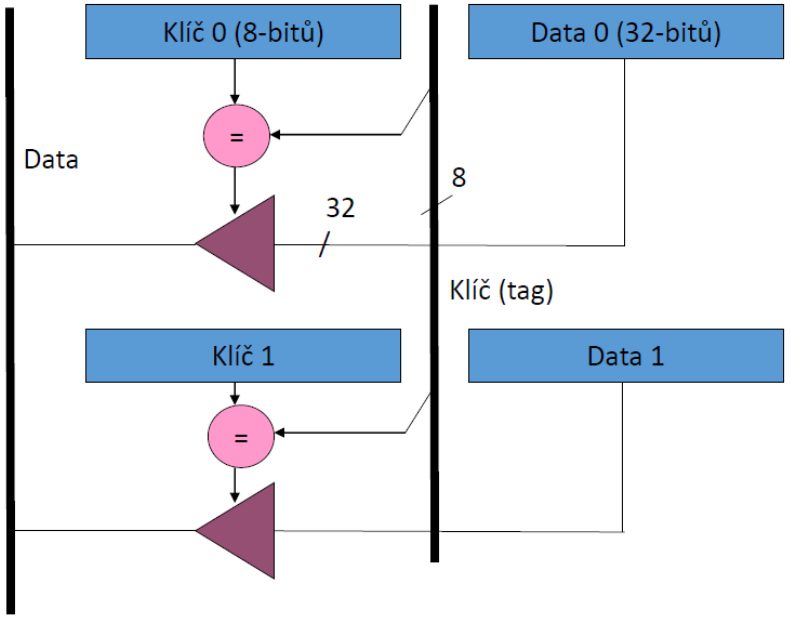
Plně asociativní cache

Plně asociativní cache

- Celá adresa brána jako tag (klíč) **více komparátorů**
- **Nepotřebuje adresový dekodér**

V praxi se používá 2. a 4. stupeň asociativity

- **Hit time** – čas ke zjištění přítomnosti dat v cache
- **Miss penalty** – čas načtení bloku ze vzdálenější paměti



Operace s pamětí cache

- Při **čtení** dat z paměti cache se data **pouze zkopírují** a cache zůstane nezměněná
- Při **zápisu** do paměti cache je nutné zajistit **konzistenci** dat

Strategie zápisu do paměti cache:

- **Write-through** – aktualizuje se položka v cache a zároveň v hlavní paměti
- **Write-back** – aktualizuje se položka pouze v cache, zápis do hlavní paměti nastane až při přepsání bloku v cache

Obě strategie využívají **write buffer**, který umožní rychlejší zápis a nemusí se čekat na zpoždění při zápisu do hlavní paměti

Příčiny cache miss

- **Miss rate** – počet přístupů, které chybí (cache miss) vydělené celkovým počtem přístupů

Příčiny cache miss:

- **Povinný přístup** (Compulsory) – blok při prvním použití dat není v cache, musí být načten do paměti cache. Nastane, i kdyby cache byla nekonečně velká.
- **Kapacita cache** (Capacity) – cache neobsáhne všechny bloky při provádění programu, musí být vyřazeny a později zase načteny
- **Konflikt** (Conflict) – pokud cache není plně asociativní, může dojít ke cache miss (kromě předchozích příčin) z důvodu mapování více bloků do téže sady

Optimalizace cache I.

1. **Zvětšení velikosti bloku** – redukuje compulsory miss, zvyšuje miss penalty, závisí na velikosti cache, snižuje počet tagů
2. **Větší kapacita cache** – redukuje miss rate, prodlužuje hit time, navyšuje cenu a ztrátový výkon
3. **Navýšení stupně asociativity** – redukuje miss rate, snižuje konflikty, prodlužuje hit time, navyšuje ztrátový výkon
4. **Víceúrovňové cache** – redukuje miss penalty, rozložení úrovní urychluje čas ukládání do mezipaměti, udržuje vysokou frekvenci danou procesorem nebo vytváří velkou mezipaměť pro zajištění přístupu z procesoru do hlavní paměti, jsou výkonnější než jedna agregovaná vyrovnávací paměť

Optimalizace cache II.

5. Upřednostnění **priority čtení** při ukládání hodnoty z cache do hlavní paměti přes **write buffer** - tato hodnota je znovu čtena před dokončením uložení. Může vzniknout **nekoherence** dat, poněvadž cache obsahuje aktualizovanou hodnotu místa v hlavní paměti potřebného pro čtení, která do ní ještě nebyla uložena.
6. **Vyhnutí se překladu adresy** během indexace cache – při překladu virtuální adresy z procesoru na fyzickou adresu pro přístup k paměti je využit **offset stránky identický v obou adresách** (virtuální i fyzické). Metoda virtuálních indexů/fyzických tagů vede k omezení velikosti a struktury mezipaměti L1, přesto je výhodné vyjmutí TLB (translation lookaside buffer) při překladu.

Way prediction

- Používá **extra adresový bit** (predictor) k předvídání cesty nebo bloku v sadě pro další přístup do cache. Vybírá, které bloky vyzkouší pro další přístup do paměti cache
- **Multiplexor může být nastaven dříve**, aby se vybral požadovaný blok a mohl se **současně číst**
- Pokud predictor vybere vhodný blok, zpoždění při přístupu do cache je minimální – **fast hit time**
- V případě cache miss je testován v dalším hodinovém cyklu jiný blok

Sřetězení přístupu do cache

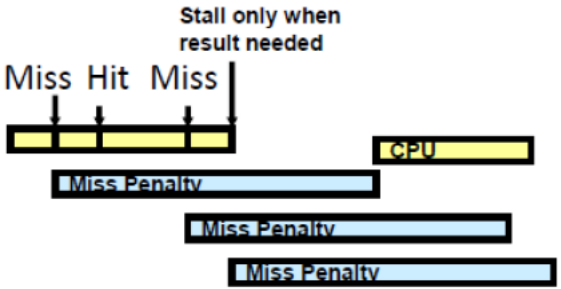
- Sřetězení přístupu do cache – L1 cache hit může probíhat několik hodinových cyklů při vysoké propustnosti cache
- Využíváno při předvídání načítaných instrukcí a dat, např cykly

Non-blocking cache

- Umožňuje další cache hit během cache miss
- Snižuje průměrnou miss penalty
- Dochází k provedení mimo pořadí
- Výrazně zvyšuje složitost řadiče cache (více nevyřízených přístupů najednou)
- Vyžaduje sřetězený systém

Stall CPU on miss

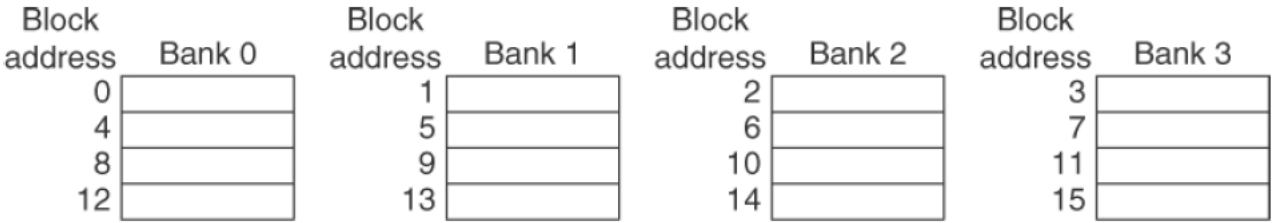
Hit under miss



Multiple out-standing misses

Vícenásobné banky cache

- Je vhodné rozdělit paměť cache na banky s nezávislým přístupem
- Jednoduché mapování – postupné prokládání, např. pro 4 banky má banka 0 všechny bloky, jejichž adresa modulo 4 je 0, banka 1 má bloky, jejichž adresa modulo 4 je 1



Snížení Miss penalty I.

- Založeno na principu, kdy procesor **potřebuje právě jedno slovo bloku** v čase, nečeká na načtení celého bloku před zasláním požadovaného slova
- Předčasný restart – při načtení požadovaného slova v bloku **procesor pokračuje** ve vykonávání instrukcí
- Nejprve jsou načtena chybějící slova z paměti a odeslány do procesoru, následně procesor pokračuje ve zpracování ostatních požadovaných slov bloku

Snížení Miss penalty II.

Merging Write Buffer

- Write buffer umožňuje procesoru pokračovat zatímco se čeká na zápis z cache do paměti
- Pokud buffer obsahuje modifikované bloky, adresy lze zkontrolovat, jestli **adresa nových dat odpovídá adrese platné položky vyrovnávací paměti pro zápis**
- Pokud tomu tak je, nová **data jsou spojena s danou položkou**
- Zvyšuje velikost bloku pro zápis write-through cache při sekvenčním zápisu slov

Optimalizace překladu

- **Struktura kódu** ovlivňuje **sled přístupu k datovým blokům** (skupinová data – prostorová lokalita, změna pořadí přístupu ke zlepšení časové lokality)
- **Zákaz přístupu k datům přes cache** (vhodné pro **proměnné použité pouze jednou**, využívá mechanismus sdělení mezi SW a HW pro překlad - instruction hints nebo page table bity)
- **Zahození dat, které se nikdy znovu nepoužijí** (využívají prostorovou lokalitu, jsou nahrazeny mrtvá místa v cache)

Výměna pořadí ve smyčce

- Výměna smyčky (loop interchange) zajistí **zlepšení prostorové lokality**, přičemž přeskupení maximalizuje využití dat v bloku vyrovnávací paměti před jejich vyřazením.

```
for(j=0; j < N; j++) {
    for(i=0; i < M; i++) {
        x[i][j] = 2 * x[i][j];
    }
}
```

```
for(i=0; i < M; i++) {
    for(j=0; j < N; j++) {
        x[i][j] = 2 * x[i][j];
    }
}
```

Sloučení smyčky

- Sloučení smyčky (loop fusion) zajistí **zlepšení časové lokality**, přičemž přeskupení maximalizuje využití dat v po sobě vykonávaných instrukcích.

```
for(i=0; i < N; i++)
    for(j=0; j < M; j++)
       a[i][j] = b[i][j] * c[i][j];

for(i=0; i < N; i++)
    for(j=0; j < M; j++)
       d[i][j] = a[i][j] * c[i][j];
```

```
for(i=0; i < M; i++)
    for(j=0; j < N; j++) {
       a[i][j] = b[i][j] * c[i][j];
       d[i][j] = a[i][j] * c[i][j];
    }
```

Blokování

- Tato optimalizace **zlepšuje časovou lokalitu** ke snížení cache miss.
- Při práci s poli při ortogonálním přístupu (kdy se v každé iteraci použijí řádky i sloupce) se využívají **blokované algoritmy pracující na submatricích** nebo blocích. Cílem je maximalizovat přístup k údajům načteným do vyrovnávací paměti před výměnou dat.

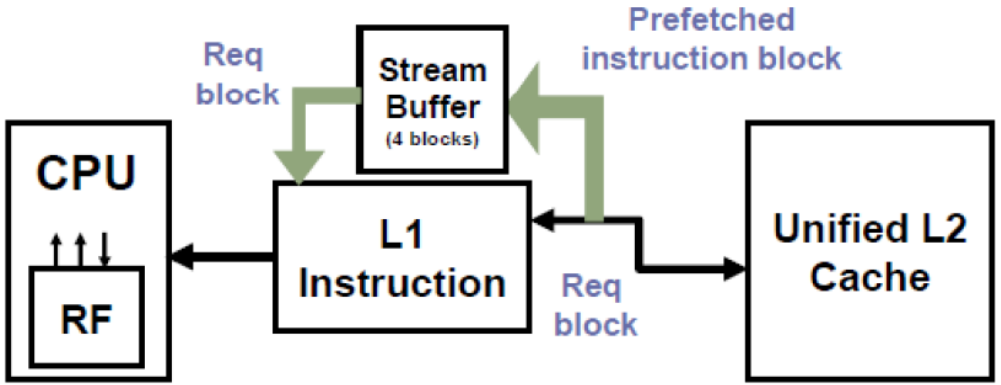
Prefetching - předvídání.

- **Předběžné načtení** položek do cache před tím, než o ně procesor požádá

- Při cache miss procesor načte dva bloky: **požadovaný blok a následující**.
- Závisí na **propustnosti cache** – předběžné načtení dalšího bloku, který se pak nevyužije, snižuje výkon. Pokud předvídání funguje správně, je dopad na výkon zanedbatelný
- **Předvídání instrukcí je snadnější, než předvídání dat**

Hardwarové předvídání

- Požadovaný blok je umístěn do instrukční cache a předem načtený blok je umístěn do **stream bufferu**.
- Pokud je následující požadovaný blok **přítomen v bufferu**, původní **žádost** o načtení do cach je **zrušena**, blok je čten z bufferu a je vydán další požadavek předběžného načtení.



Předvídání řízené překladačem

Kompilátor **vloží instrukce pro předběžné načítání**. Možnosti:

- **Register prefetch** – hodnota načtena do registru
- **Cache prefetch** – hodnota načtena pouze do mezipaměti – více používané

Obě možnosti mohou být úspěšné nebo neúspěšné, v případě nedovoleného čtení **není ale vyvolána vyjímka** pro chyby virtuálních adres a porušení ochrany, poněvadž nepoužitelná instrukce je nahrazena instrukcí no-operation.

- Předběžné načtení instrukcí má smysl pouze ve spojení s předběžným načtení dat
- Mezipaměť poskytuje instrukce a data bez blokování, **u předem načtených dat čeká**, až se vrátí **po zpracování**
- Nejefektivnější předvídání je sémanticky neviditelné pro program: Nezmění obsah registrů a paměti a nemůže způsobit poruchy virtuální paměti

Softwarové nástroje

PREFETCH adr	- nahraje obsah hlavní paměti z adresy adr do skryté paměti
CLFLUSH adr	- zneplatní data z adresy adr ve skryté paměti (následuje-li čtení z adr, pak se vždy data načtou z hlavní paměti)
INVD	- zneplatni celý obsah skryté paměti
WBINVD	- zapiš obsah skryté paměti do hlavní paměti a zneplatni celý obsah skryté paměti

PREFETCH a CLFLUSH se používají při zpracování větších objemů dat. Dává se jimi paměťovému systému předem na vědomí, že určitá data budou brzy potřeba a naopak (tj. nebudou již potřeba).

Příklad software prefetching

```
for(i=0; i < N; i++) {
    prefetch( &a[i + 1] );
    prefetch( &b[i + 1] );
    SUM = SUM + a[i] * b[i];
}
```

- Největším problémem je vhodné načasování načtení
- Načtení dat těsně před jejich použitím - může být již pozdě
- Načtení dat moc brzy – nekoherence
- Můžeme nastavit parametr P (místo +1) k vhodnému odhadu načasování
- Cykly je možné optimalizovat předem
- Při malé miss penalty rozpracuje dva kroky předem, jinak použije pipelining