1.6 Cache memory

Monday, March 23, 2020 2:42 AM

I hver instruksjonssyklus aksesserer prosessoren minne minst én gang, til å fetche en instruksjon, og ofte flere ganger for å fetche operander og/eller lagre resultater. Som følge av Moore's lov kan en prosessor execute instruksjoner langt raskere enn minnes sykeltid (cycle time). Ideelt burde hovedminneteknologien vært bygget med samme teknologi som prosessorens registre, slik at minnes sykeltid kunne vært sammenlignbart med prosessorens sykeltid. Likevel har dette bevist seg å være en altfor dyr strategi, og derfor brukes lokalitetsprinsippet til å løse dette problemet ved bruk av cache.

Cache-prinsipper

Cacheminne er ment å gi en minneaksesstid som nærmer seg den av de raskeste minnene tilgjengelig, og samtidig støtte en større minneplass som har samme pris som mindre dyre typer av semiconductor minner. Cache inneholder små deler av hovedminne. Når en prosessor forsøker å lese av en byte eller et ord fra minne sjekker den for om byten eller ordet er i cache. Hvis den befinner seg i cache blir byten eller ordet levert til prosessoren. Dersom den ikke befinner seg i cache lastes det inn en block (bestående av et gitt antall bytes) til cache før den så når prosessoren. Når en block er lastet inn i cache for å tilfredsstille en enkel minnereferanse, så følger det av lokalitetsprinsippet at det er sannsynlig at mange av minnereferansene som skal brukes i nærmeste fremtid vil være andre bytes i blocken som ble lastet inn til cache.

Def. *Ord*	En benevnelse som benyttes om den naturlige datastørrelsen i en gitt datamaskin. Et ord er ganske enkelt en samling av bit som håndteres samtidig av maskinen. Antallet bit i et ord, kalt ordstørrelsen, er et viktig kjennetegn ved enhver datamaskinarkitektur. Vanlige ordstørrelser i dagens PC-er er 32 og 64 bit. De fleste registrene i en datamaskin har størrelsen til et ord.

Struktur

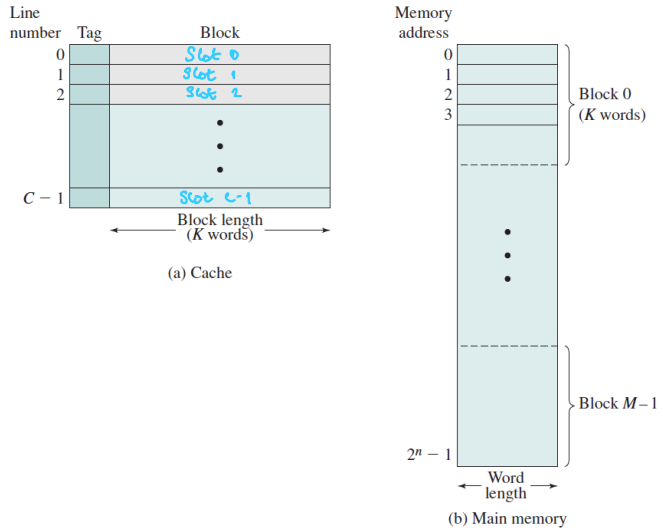


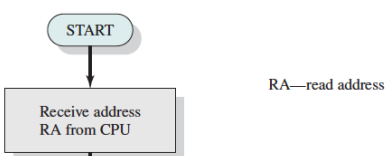
Figure 1.17 Cache/Main Memory Structure

Hovedminnet består av oppimot 2^n adresserbare ord, der hver ord har en unik *n*-bit adresse.
For mapping-grunner er dette minnet betraktet å inneholde et antall fast lengde blokker (blocks) av *K* ord hver. Det vil si at det er $M = 2^n / K$ blokker. Cache består av *C* slots (også kalt linjer/lines) av *K* ord hver, og antallet slots er betraktelig mindre enn antallet blokker i hovedminnet ($C << M$). Hvis et ord i en blokk i hovedminnet ikke blir lest fra cache blir den blokken overført til slot i cache. Fordi det er flere blokker enn slots, kan ikke en enkel slot være unik og permanent dedikert til en bestemt blokk. Derfor inneholder hver slot en tagg som identifiserer hvilken blokk som blir lagret. Denne taggen består normalt av høyere-ordens bits fra adressen, og refererer til alle adressene som begynner med denne sekvensen av bits.

Eks. Anta at vi har en 6-bit adresse og en 2-bit tagg. I så fall vil taggen 01 referere til blokken med lokasjoner til adressene med sekvensen 01 i de høyere-ordens bit-ene:

010000, 010001, 010010, 010011, 010100, 0101010, 010110, 010111, 011000, 011001, 011010, 011011, 011100, 011101, 011110, 011111.

Figuren under demonstrerer en read-operasjon med cache.



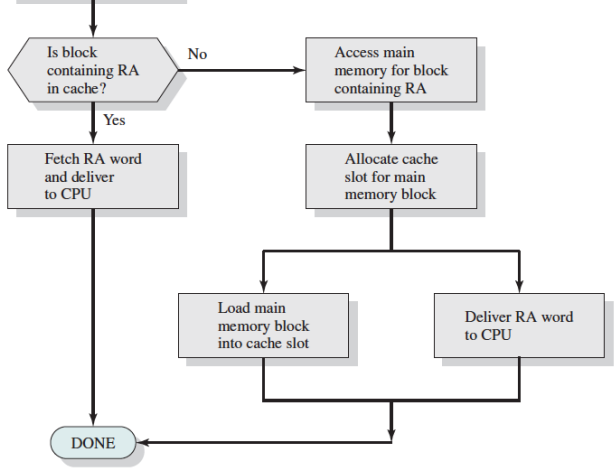


Figure 1.18 **Cache Read Operation**

Cache design

Det er noen hensyn man må ta ved design av cache.

- **Cache størrelse**: For stor cache betyr større kontroll blokker, som betyr at det tar mer tid å søke frem til riktig cache slot. Det vil si at en trenger flere klokkesykler for å fetche en cache slot, som motstrider hele konseptet med cache (rask aksesstid).
- **Blokk størrelse**: Størrelsen på data som sendes mellom hovedminne og cache. Hvis man øker blokkstørrelsen for mye mister man poenget med lokalitetsprinsippet, ettersom hit ratio vil bli mindre da det er for mye data som blir lastet inn som ikke er relevant i nærmeste fremtid.
- **Mapping funksjon**: Når en ny blokk av data blir lest inn i cache er det mapping-funksjonen som avgjør hvilken cache lokasjon blokken vil oppta. To begrensninger påvirker designet av mapping-funksjonen. Når en blokk lastes inn i cache kan det hende at den må erstatte en annen. Vi ønsker å maksimere hit ratio, så en fleksibel mapping funksjon har større kjangs. Likevel er det slik at en mer fleksibel funksjon, har desto mer kompleks infrastruktur og dermed er vanskeligere å søke på om blokken finnes i cache.
- **Erstatningsalgoritme** (Replacement algorithm): Har som oppgave å velge hvilken blokk som skal erstattes. En strategi er å velge den blokken som har vært lengst i cache uten en referanse, kalt LRU-algoritmen (least-recently-used). Hardwaremekanismer identifiserer hvilken blokk som har vært inne lengst.
- **Skrive-retningslinjer** (write policy): Hvis innholdet i en blokk i cache får innholdet sitt endret er det nødvendig å skrive tilbake til hovedminne før man erstatter. *Write policy* dikterer når skriveoperasjonen finner sted. I de mest ekstreme tilfellene kan skrivingen oppstå hver gang en blokk blir oppdatert, eller bare når en blokk blir erstattet. Sistnevnte minimerer skriveoperasjoner til minne, men etterlater hovedminne i en overflødig tilstand. Dette kan få konsekvenser for flerprosessor(multi-processor)-operasjoner og direkte minneaksesser av I/O-hardware-moduler.
- **Antall nivåer av cache**: Det er i dag vanlig med flere nivåer av caches, vanligvis kalt L1 (nærmest prosessoren->), L2, L3, osv.