Paměťový subsystém

25. listopadu 2008
11:20

Parametry paměti

- Kapacita
  - Objem informace, který je možno uchovat v jedné paměťové jednotce (obvykle ve slovech nebo bytech) - kde víme kolik bitů je jedno slovo
  -
- Velikost slova
  - Velikost přirozené jednotky paměti
  - (obvykle počet bitů, pomocí kterého je slovo reprezentováno)
- Přenosová jednotka
  - Počet dat, kolik přenášíme v 1 kroku
    - Pokud jde o hlavní paměť, tak udáváme v bitech
    - U sekundární paměti o počtu bloků
- Přenosová rychlost
  - Rychlost, kterou mohu přenášet data z/do paměti (maximální vs. Zaručená)
- Vybavovací doba
  - Čas za který je paměť schopna vyřídit požadavek = doba přípravy na požadavek
- Cyklus paměti
  - Doma mezi dvěma bezprostředně za sebou jdoucími požadavky
- Přístupová metoda

Podle čeho se dají paměti dělit

- Podle funkce
  - Jen čtení, nebo i zápis?
- Způsob přístupu
  - Sekvenční, nebo kdekoliv
- Technologie
- Umístění v sytému
  - Registry, hlavní paměť, archivní ve formě CD
- Vnitřní organizace
  - Jestli uvnitř je další logická struktura, nebo pole indexované adresami?
- Detekce / opravy chyb
  - Jestli spoléhá jen na ukládání dat, nebo i hlídá sama? (CRC checksumy např.)

- Dřív paměti byly rychlejší jak procesor, dnes je tomu naopak

- Podle funkce
  - ROM - read only memory
  - RWM - read write memory
  - *Speciální - IRAM (Intelligent ram); CRAM (Crypting RAM);*
  - WORM - write once, read many

  Varianty
  - ROM
    - Mask ROM - obsah je dán přímo maskou při výrobě - neflexibilní, změna jen změnou výroby
    - PROM - programable read only memory (naprogramuji jednou, pak uz nezměním)
    - EPROM - eraseabel programmable memory (dříve pro ukládání firmwaru) smazání např. osvícením ultrafialovým světlem; prakticky se muselo místo pro smazání zalepovat, pokud nálepka odpadla, tak se data mohly smazat
    - EEPROM - electronicly eraseable programable memory - všude kolem nás :-)
    - Flash

      Dost často jsou rozdíly ne ve výrobě, ale na využití.. EPROM = PROM (jen nebyl prostor pro okénko na smazání)

- RWM
  - DRAM - dynamická - po čase obsah zapomínají a obsah je třeba udržovat
  - SRAM - statická - udržují si svůj obsah po celou dobu, co mají napájení

- Podle způsobu přístupu
  - RAM - random access memory [vnitřní paměti]
    - Všechna paměťová místa mají svou adresu
    - K paměťovým místům může být přistupování v libovolném pořadí
    - Doba přístupu nezáleží na předchozí adrese, je konstantní
  - SAM - sekvenční access memory [páskové zálohovací paměti]
    - Pamětova nemusejí mít svou adresu, ale je dáno vzdáleností od začátku
    - Sekvenční přístup
    - Doba přístupu je závislá na vzdálenosti od počátku
    - V dnešních počítačích je použití pro videopamět
  - DAM - direct access memory
    - Paměťová místa mají jednoznačné adresy
    - Např. zvolím si oblast paměti, a tam to pak už projdu sekvenčně
    - Typické pro disky .. Oblast je sektor - a zbytek přečtu sekvenčně dokud se nedostanu ke svým datům
    - Není konstantní ani lineární
  - AAM - asociative access memory
    - Podle části oblasti
    - Telefonní seznam např. hledám podle části obsahu
    - Např. v cache kde mám data a adresu kde data původně byly
    - ¨používá se paralelní prohledávání, které je ale implementačně drahé

  - Technologie paměti
    - Preelektornické - relé,, zpoždovací LINKY, FERITOVÁ POLE
    - Elektronické - RAM, FLASH
    - Magnetické - bubny, pásky, disky
    - Optické - CD, DVD
    - (chemické, biologické...)

      Elektronické
      - Statické
        - ◇ Pro udržení dat není třeba periodicky obnovovat
        - ◇ Jak vypadá.. Viz. Slide
          - ► Základní idea je, vstupem překlopím pin, tím přepíšu a pak držím dokud mám napájení
          - ► Klopný obvod ^^ . Pomocí hradel a tranzistorů
        - ◇ Výhoda, že si pamatují data dokud napájí.. Můžeme počítač provozovat libovolně rychle (a nemusíme se starat o obnovování)
        - ◇ Dají se taktovat i ručně
        - ◇ Pomocí jednoduchých bitových pamětí můžu složit větší pole (registrové pole)
        - ◇ SRAM . Menší počet tranzistorů - založeno na invertování
        - ◇ Možnost maticového uspořádání
        - ◇ Kapacita - výška (počet slov) * šířka (v bitech)
      - Dynamické
        - ◇ Může být založena na kondenzátorech => zapomínání
        - ◇ Destruktivní čtení
        - ◇ Limitovaná doba uložení - náboj je malý, u malých kondenzátorů
        - ◇ Malý počet součástek

        Máme adresový a bitový vodič, pomocí jednoho kondenz. A tranz.

        Zápis: data dáme na adresový vodič, dáme sepnout a kondenzátor se nabije bez ohledu co tam bylo

Čtení: nabudí … [někde najít!!!]

DRAM je obohacena o řídící logiku, která zařídí aby vše šlo se správným taktováním a ve správném pořadí

Dáme řádkovou adresu, tikneme RAS, ta zařídí výběr řádku
Pak sloupcovou část . Tikneme CAS , ta zařídí výběr řádku
A pak máme signál WE pro write enable

Zvyšování výkonu DRAM

- Nibble mode access
  - Využita lokalita přístupu
  - Po vystavení dat možno pulsy ~CAS získat 4 po sobě jdoucí data
- Page Mode Access
  - Signál RAS podržen => data držena sense amps
  - Signál CAS a sloupcová adresa měněny podle potřeby
- Fast page mode access
  - Podobné PM
  - ~RAS není držen po celou dobu => snížená spotřeba
- HyperPage Mode = Extended DataOt= EDO
  - Datový výstup může být zachován i přizměně adresy
- Burst EDO
  - Šlo to po více balíkách
  - Neujalo se to
- Video DRAm = VRAM
  - Vnitřní implementace stejná, ale přidán další interface, který obsahovall vystavený řádek
  - Pro zrychlení vykreslování

Synchronní / asynchronní přístup

- Doteď asynchronní
- Viz slide..

Synchronní přístup k paměti

- JEDEC SDRAM (PC66 SDRAM)
  - Přidání SPD chipu pro identifikaci - jaké parametry bude mít paměť
    - ◇ Přístup k němu podle jednoduchého protokolu.. Nejspíš i2c
    - ◇ Taktovací frekvence odvozena od frekvence sběrnice
- Double Data Rate = DDR SDRAM
  - Výstup aktivován jak na náběžné, tak na sestupné hraně hodinového signálu - čímž došlo k zrychlení
- Rambus DRAM - Rambus a Intel
  - Neujala se
- DDR2
  - Až 800MHz
  - Burst oriented - na balíčky dat 4 nebo 8
  - CAS latency - Collum access … - mezi jednotlivýma tikama potřebujeme nějaký čas.. Tak tady se říká kolik tiků to musí být
    - ◇ Ale tak, aby paměť stihla požadavek vyřídit
- DDR3 snížení napájecího napětí

RDRAM - rychle poskytují malá data
SDRAM - poskytují velké objemy dat, ale pomaleji

Původně se odvozovalo od taktovací frekvence, ale přečarovalo se to na… viz slide.

Paměti založené na orientovaném protokolu (tedy ne podle adresy)
- Rambus DRAM
- SyncLink DRAM (SLDRAM)

Hierarchie paměti

Registry - *procesor*
Vyrovnávací paměť - *L1, L2 cache*
Hlavní paměť - *RAM*

Odkládací sekundární paměť - p*evný dis*k
Archivní paměť - *CD, DVD*

K procesoru bude nejrychlejší, ale s malou velikostí

Vnitřní
vnější

Typ	Technologie	Velikost	Přístupová doba
Registry	Polovodiče, na procesoru	B	~ 1 ns (adekvátně k rychlosti procesoru)
Cache	Polovodiče, na/vně proc.	kB	~ 10 ns.
Hlavná p.	polovodiče	MB	10-100 ns.
Sekundární	Magnetický záznam	GB	

Vyrovnávací paměti - Cache
- Obvyklé použití
  - Procesor - hlavní paměť
  - Počítač - pomalá periferie
- Využití lokality přístupu:
  - Když procesos používá data z nějaké paměti, tak je asi bude používat znovu, tak si je předpřipraví, kdyby je opravdu potřeboval - nebude na ně muset čekat
- Organizace cache
  - Data v cache uložena se svou adresou - zaznamenání jaká data a odkud jsou vzaté; při vyhledávání kontroluju adresy jestli jsou v cache, a pokud ne, tak pak jdu do paměti (není lineární prohledávání)
- Fully associative mapping
  - Vyžaduji data z konkrétní adresy
  - Něco prohledá paralelně všechny záznamy, nevyýhodou je velká cena za prohledávání
    - Data nejsou samotná, ještě tam jsou další příznaky (napři. Jestli data ještě platí)
- Direct mapping
  - Ústupek od předchozího
    - Část adresy použiju na to, kde by asi mohly v cache být

Tag	Line [zde se říká, kde by se ty data nacházely]	word

    - Jde o to, aby používané adresy nepadaly na stejné místo (aby se nepřepisovaly)
- Set aociative mapping
  - Cache rozdělím na oblasti a každá oblast má direct mapping; to řeší ten předchozí problém přepisování

- Uvolňovací mechanismy
  - Problém: jakou potřeba data v cache není místo
  - Řešení
    - Direct mapping: jednoznačně určeno
    - LRU - naposledy použitý zůstane
    - FIFO - klasická fronta
    - LFU - data která používám nejméně
    - Random
- Konzistence dat (když někdo změní hlavní paměť a jiná cache v cpu)

- Strategie zápisu:
  - Write-through - současný zápis změny do cache i do hlavní paměti
  - Write-back - kdy data procesor mění, tak se udržují jen v cache, a do paměti se uloží až když data z cache odstraňuji
    - Problém s pamětí při sdílení (může se stát, že dva procesory si něco přečtou, a pak záleží kdo je dřív uloží) - řízení protokolem MESI - vyjadřují stavy dat v paměti a procesory můžou komunikovat navzájem
  - Write-Once
    - Při prvním zápisu se zapíšou jako write-trought pak nic a na konci při vymazání jako writeback
    - Tím řeknu, že vlastně tu paměť nějak měním

Přidělování pamětí

- V prostředí kde je více procesů:
  - Nutno řešit relokace a ochranu
  - Pevné oddíly - volné oddíly
    - Pevné: paměť rozdělena na oddíly, procesy ve frontách podle požadované velikosti (víc nedostane)
    - Volné: procesy dostávají paměť podle potřeby
      - Musím na straně systému zajistit, aby se vše dávalo správně, aby vše bylo k dispozici pro toho kdo má nárok
  - Potřeba vědět, jaká část paměti je opravdu volná
    - Bitová mapa (bloky stejné délky, "paragrafy" - adresa se rozdělí na nějaké dvě části, jedna říká uvnitř paragrafu, druhá který paragraf to je)
    - Spojový seznam bloků - vlastně říká jak je ta paměť poskládaná, adresa bloku a jeho velikost, ukazatel na další volný blok
    - Při uvolňování musím kontrolovat, jestli nejsou okolo také volné, abych případně sloučil ve větší blok
  - Strategie přidělování [problém rychlost vs. Účinnost zaplnění]
    - First fit - z prvního, do kterého se vejde přidělím co je třeba, zbytek zůstane zase volným blokem [problem, když se začne něco uvolňovat]
    - Next fit - jedu dál, kde jsem naposledy skončil, když dojdu nakonec, tak od začátku, když jednou dokola, tak takový blok tam není,
    - Best fit - problém, vznikají malinkaté kousky, které už asi nikdo chtít nebude
    - Worst fit - naopak, ukradne to z největšího volného bloku (nezůstávají ty malinké kousky)
  - Problém když hromada malých kousků řeší fragmentace
  - Virtuální paměť
    - Ochrana procesů mezi sebou samotným, relativní adresování ve vlastním prostoru
      - Nemůžu do cizího
    - Vlastně můžu dělat, že máme víc paměti než doopravdy je
      - Navenek: občas se stane, že přístup k nějaké paměti bude dlouhý (data se berou odjinud)
    - Snížená fragmentace
      - Měníme jen mapování mezi virtuální a fyzickou
    - Snížení nebezpečí uváznutí - když někdo chce víc paměti než kde, tak někoho na chvíli odstřihneme
    - Pohodlné
      - Stejný svět pro všechny; každý proces má svůj adresový prostor; nehrozí ovlivňování
      - Rozšíření prostoru - můžu působit, tak, že ukazuji různé velikosti prostorů
    - Jistá abstrakce reálné paměti
    - Klíč
      - Převod virtuálních adres na fyzické (parciální zobrazení)
    - 2 základní metody:
      - Stránkování
        - Stránka má stejnou velikost jako rámec ve fyzické paměti
        - A správce jen řeší přemapování stránka < > rámec
        - Virtuální adresový prostor rozdělený na stránky, fyzický na rámce
        - Pohled uživatele: spojitý adresový prostor (ve skutečnosti to může být

rozházené po hlavní i sekundární paměti, nebo vůbec
- Stránkování je neviditelné pro proces
- Adresu vyjádříme jako dvojici [p = stránka, posunutí]
  - Mechanismus tránkování převede číslo stránky p na odpovídající číslo rámce p' (jde-li to) - mapování
  - Adresa ve fyzickém vyjádřena [p', posunutí]
  - Neexistuje li mapování, dojde k výpadku stránky (page fault)
  - Ta jednotka může být poměrně jednoduchá, samotné promazávání a hledání řeší operační systém; dokud není tohle vyřízeno je proces pozastaven
- Segmentace

- Problémy stránkování:
  - Interní fragmentace - když potřebuju malinkou část, stejně použiju celou stránku
  - Velikost stránkovacích tabulek 32b adresy -> 4GB paměti, stránky 4kB => stránkovací tabulka má 1M položek
  - Rychlost přístupu do stránkovacích tabulek (operace masivně zacházející s pamětí stále budou přistupovat do paměti)
- Výběr velikosti stránky
  - Malé stránky
    - + menší fragmentace
    - - velká adresová tabulka
  - Velké stránky

- Úpravy
  - Víceúrovňové stránkování - řeší problém velikosti tabulek
    - Podtabulky můžu už také ukládat jinam
    - Vícenásobná virtuální
    - Adresa 3 části, directory - table - offset
  - Asociativní paměť
    - Přístup k tabulkám je rychlý, ale drahá paměť
  - Nulaúrovňové stránkování
    - Extrémní případ použití předchozího
    - Nejsou stránkovací tabulky, pouze asociativní paměť
  - Inverzní stránkovací tabulky
    - Organizace nad rámci, nikoli stránkami
    - Nevýhoda: když chci zjistit jestli je nějaká stránka v rámci, tak musím projít celý rámec (lineárně)
- Záznam stránkovací tabulky [slide]
  - P .. Present.. Říká, jestli ta stránka je
- Algoritmy výměny stránek
  - Když je plno, tak podle toho říkáme koho vyhodit
  - Optimální stránka - chtěl bych vyhodit stránku, kterou už nikdy nebudu potřebovat (nedá se moc dobře odhadnout)
  - Random - náhodný výběr
  - Working sett

[page-fault.. Zápis do stránky, která není namapovaná]
Pagefault nastává kdykoliv, kdy stránka není označena za přítomnou - to označení nastavuje OS; když dojde k pagefaultu, tak systém zkusí bud na disku načíst, nebo když leze proces někam, kam nemůže, tak to zabije

Procesy si můžou povídat

I kdyby úsek paměti mel 3 stránky, tak může zabrat 4 stránky