Adresování paměti

T.Mainzer

Adresní prostor

Logický adresní prostor - Adresní prostor se kterým může pracovat/může adresovat daný procesor. Pracuje li procesor s 16-bitovou adresou má log.adresní prostor velikost 2^16 (64kB), pracuje li s 32-bitovou adresou je log.adresní prostor 2^32 (4GB)

Fyzický adresní prostor – Velikost skutečně osazené (fyzicky přítomné a adresovatelné) paměti.

Pozn: Kromě paměťového adresního prostoru může být ješte I/O (V/V) adresový prostor příp. prostor pro další specifické fce

Velikost logického (L) a fyzického (F) adresního prostoru	
L=F	Nic neřešíme
L>F	Typický stav, řeší se mapování paměti (mapování logické adresy na fyzickou). Někdy je mapování pevně dané (typicky mikrokontrolery). U procesorů mapování často zahrnuje i mechanismus virtuální paměti (automaticky řešeno HW)
L<F	Netypický případ, lze to, ale přístup do paměti musí být řešen na úrovni SW i na aplikační úrovni

Adresní módy (v instrukcích)

(Přímý operand – hodnota/číslo uvedena v kódu instrukce)

Přímé absolutní adresování – přímo adresa/číslo odkazující na pamět (A)

Adresování registrem - Nepřímé adresování, (R) registr obsahuje adresu

Bázové s posunem – (A+R) – Přímá adresa + posun v registru

Nepřímé s offsetem – (R+I) – Nepřímá adresa v registru + offset I

Indexované – (R+R) – Adresa i offset v registrech

Indexovaný s měřítkem - (R+R*m)

Nepřímé přez paměť – ((R)) – registr obsahuje adresu na které je ukazatel

Autoinkrement/Autoderement – (R++, R++m) čtení z paměti a navýšení/snížení ukazatele

Bitově reversibilní adresování – spodní část adresy je bitově otočena, výpočet FFT

Kruhové adresování – po dosažení konce oblasti se pokračuje opět na začátku

Segmentace a Stránkování

Segment = Virtuální adresní prostor různé velikosti (=segment) začínající od (virtuální) adresní nuly s aplikačně různým určením (tj. rozdělení na segmenty respektuje log.strukturu programu/dat/OS) – typicky segment kódový, segment (statických) dat, segment zásobníku, segment systémový atp. Každý segment může mít různá přístupová práva (kódový segment jen pro spouštění, do zásobníkového segmentu může přistupovat jen jeden proces atp.).

Výhody: Ochrana přístupu procesů (práva), velikost segmentu je uzpůsobena potřebě, lze měnit umístění segmentu v paměti jen změnou segmentu, offset v segmentu zůstává zachován

Nevýhody: Obecně problémy s alokací segmentů při změnách velikosti a možná fragmentace, režie při přístupu do paměti (převod adresy přes tab.segmentů)

(Algorimy přidělování paměti / volného bloku – Firstfit,lastfit, worstfit, exactorworstfit, bestfit)

Stránka = virtuální adresní prostor je rozdělen na stránky stejné velikosti. Jejich reálné umístění ve fyzické paměti je dáno překladem (tab.stránek, překládá log.adresu na fyzickou)

Výhody: omezení fragmentace paměti, možnost přístupových práv

Nevýhody: Nedokonalé využití paměti (dané velikostí stránky), režie při přístupu do paměti (převod adresu přes tab.stránek, často víceúrovňovou)

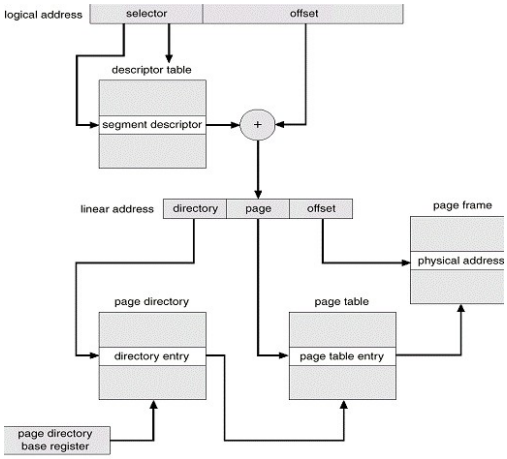
(Algoritmy výběru stráky pro vyhození/oběti - optimal(teoreticke),fifo,odložené fifo, LRU (lastrecentlyused), LFU (lastfrequentlyused), pseudo-LRU)

pozn. Jak pro segmentování tak pro stránkování lze použít mapování, kdy ve fyzické paměti je jen část programu/dat a ty části které se nepotřebují jsou odloženy na jinde (typicky na disku) a při jejich potřebě se do paměti přesouvají (výpadek segmentu, swapování)

Segmentace+Stránkování = kombinace výše uvedených metod. Ponechává výhody segmentace (první krok), a díky stránkování (druhý krok) řeší problém s fragmentací a umožňuje mít v paměti jen používané části segmentu

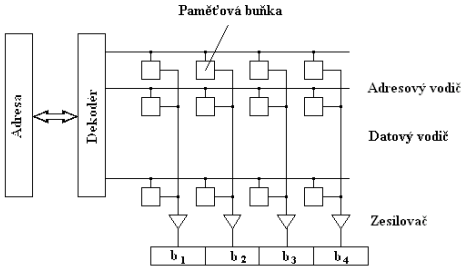
Aby výše uvedené bylo efektivní je toto nutné řešit na úrovni HW (MMU =memory management unit)

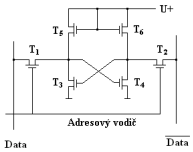
Příklad přístupu do paměti pro segmetanci + dvouúrovňové stránkování (tj. překlad log.adresy na fyzickou, logická adresa ve tvaru segment:offset je tab.segmentů převedena na lineární adresu a ta je pře dvouúrovňovou stránkovací tabulku převedena na fyzickou adresu)

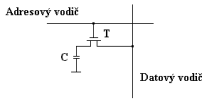


Paměť RAM - vnitřní struktura, adresace

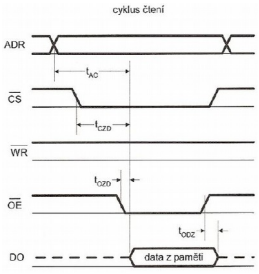
Obecná Vnitřní struktura paměti a paměťové buňky SRAM vs DRAM.

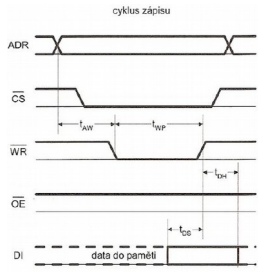






ad **SRAM** – pro uchování stavu (bitu) klopný obvod (4T (ECL) nebo 6T (CMOS), T=tranzistorů) (odtud S=statické), buňka je rychlá, ale prostorově (=cenově) náročná. Čtení je nedestruktivní (na rozdíl od DRAM).





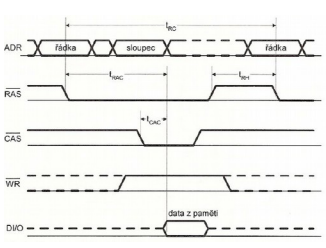
ad **DRAM** – pro uchování stavu (bitu) kondenzátor (velmi nízké kapacity <<1pF), čtením se kondezátor vybije a je nutné informaci obnovit. Kvůli samovolnému vybíjení je nutno informaci v paměti pravidelně obnovovat (odtud D =dynamické, cca 10-ky ms). Toto obvykle řeší externí obvody. Adresování DRAM je dvoustupňové řádek (R) a sloupec (C), pro refresh stačí řádky.

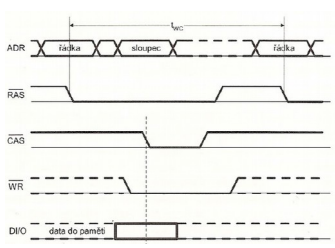
Čtení z DRAM je relativně pomalé (10-ky ns). Kvůli nízké kapacitě buňky je uchovávaná informace a čtecí/zápisová elektronika citlivá (používají se diferenciální čtecí zesilovače) např. na napájecí napětí, teplotu, časování atp. Také vzhledem k nízké kapacitě/malému rozměru může být údaj v buňce poškozen měkkým zářením (ionizace izolační vrstvy a tím ztráta náboje v buňce). Proto se používají korekční ECC kódy.

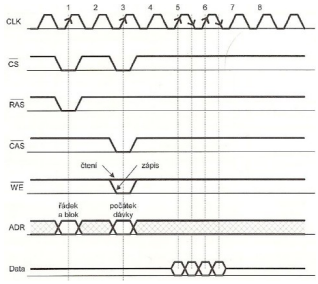
SDRAM = synchronní DRAM (všechny signnály jsou sychronizované s hodinami)

DDR = SDRAM se synchronizací na obě hrany hodin (double data rate)

čtecí a zápisový cyklus pro DRAM + cyklus pro DDR







Sběrnice, Adresní dekoder

- Sběrnice Datová, adresová, řídící(kontrolní)
- Datová – šířka dat (8bitobá, 16bitová, 32bitová,…)
- Adresová – velikost paměti (16bitů = 2^16 = 65536 adres)
- Řídící – činnosti paměti, typicky (může se lišit):
  - CS = Chipselect – povolení komunikace s pamětí (0/1=zakázáno/povoleno)
  - /CS = neg.Chipselect – povolení komunikace s pamětí (0=povoleno/zakázáno)
  - OE = Output enable – povolení výstupu na datovou sběrnici (čtení)
  - /WE = write enable – povel/řízení zápisu do paměti

Dekódodání adres

Fyzická paměť (fyzický adr.prostor) je typicky menší než adresovatelná paměť (logický adr.prostor)

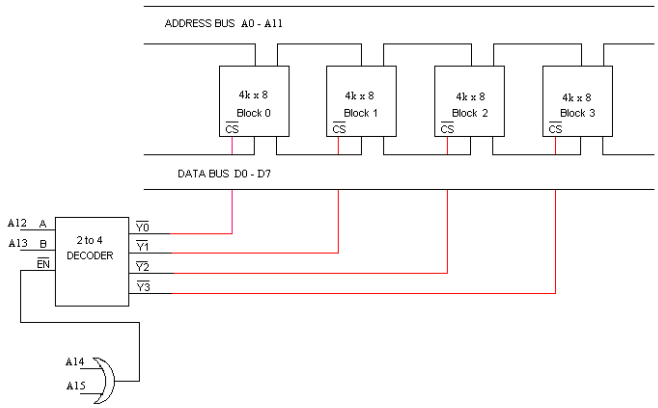
Např.

- Fyzická paměť 8bitů adres.zběrnice (A0 az A7) = 256Byte = rozsah adres 00-ff

- procesor má 10bitovou adr.sběrnici (A0 az A9) = 1kB = rozsah adress 000-3ff

Procesor-Dekoder-Pamet	Mapování (co/odkud čteme z adres 000 až 3ff)
A0-A7 ↔ Pameť A8-A9 nezapojeno	Základní nejednodušší, log.prostor je mapován: 0 1 2 3 .. FF 0 1 2 3 .. FF 0 1 2 3 .. FF 0 1 2 3 .. FF
A0-A7 ↔ Pameť A7..A0 A8-A9 nezapojeno	Prohozené adresová bity, log.prostor je mapován: 0 80 40 C0 .. 7F FF 0 80 40 … FF 0 80 40 … FF 0 80 40 … FF
A0-A1 nezapojeno A2-A9	Nepsrávné použití, log.prostor je mapovan: 0 0 0 0 1 1 1 1 2 ...
Dvě paměti X a Y A0-A7 ↔ Pameť X i Y A8 – dekoder X nebo Y A9 nezapojeno	Dekodovani adresy, log.prostor je mapovan: x0 x1 … xFF y0 y1 … yFF x0 x1 … xFF y0 y1 … yFF pokud by byl A9 zapojen do dekoderu pak: x0 x1 … xFF y0 y1 … yFF nic nic
Dvě paměti X a Y A0 – dekoder X nebo Y A1-A8 ↔ Pameť X i Y A9 nezapojeno	Dekodovani adresy, log.prostor je mapovan: x0 y0 x1 y1 … xFF yFF x0 y0 x1 y1 … xFF yFF
A0-A7 ↔ Pameť X i Y A8,A9 – dekoder nic-X-Y-nic	Dekodovani adresy, log.prostor je mapovan: nic(256B) x0 x1 … xFF y0 y1 … yFF nic(256B)

Příklad zapojení

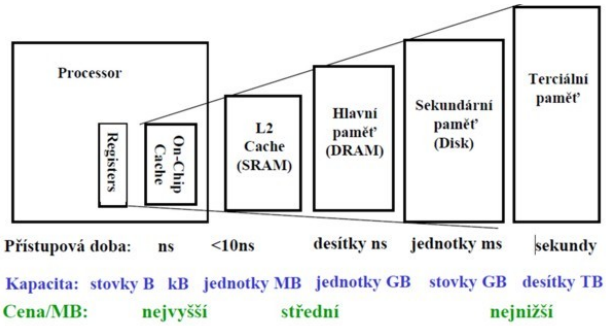


Cache

Hierarchie paměťového systému

- ideální paměť: co nejrychlejší, nejlevnější, největší, držící informace i po vypnutí

- nelze docílit splnění všech podmínek najednou, tj. Paměťový subsystém se skládá z různých druhů pamětí



Při přístupu do paměti se obvykle projevuje

časová lokalita – tj. Byla li položka použita bude pravděpodobně použita znova (proměnné v algoritmech a cyklech, instrukce v cyklu)

prostorová lokalita – tj. Byla li použita nejaká položka bude pravděpodobně použita i nějaká okolní (práce s poli, s vrcholkem zásobníku, následující instrukce)

původní počítačové systémy – rychlost paměti > rychlost procesoru. Nové systémy – rychlost paměti < rychlost procesoru → použití Cache

Cache Paměti (vyrovnávací paměti)

rychlá vyrovnávací paměť umístěná typicky mezi procesorem a hlavní pamětí , „vyrovnává“ jejich rozdílné rychlosti, může být i víceúrovňová (L1,L2). Používá se SRAM paměť (rychlejší než DRAM, ale menší kapacity → Vel.Cache << Vel.Paměti), snaha aby informace byli v Cache a co nejméně se přistupovalo do hlavní paměti. V cache je třeba rychle nalézt jsou li tam obsažena hledaná data → cache by měla být **asociativní paměť** kde klíčem je adresa do hlavní paměti – tj. ideálně v jeden okamžik porovná paralelně vstupující klíč se všemi uloženými klíči (fce XOR).

Koherence cache (zápis)

Pokud něco zapíšeme do cache je nutné to také (někdy) zapsat do hlavní paměti

Přímý zápis – po změně v cache okamžitě zapisuji i dále (→pomalé)

Zápis s mezipamětí – spec.mezipamět pro zápis omezené velikosti

Zpětný zápis – Zapisuji v okamžiku kdy přepisuji položku v Cache

Zpětný zápis změněného - Zapisuji v okamžiku kdy přepisuji položku v Cache ale jen změněné/zapsané položky (tj. je zde navíc příznak zápisu tkzvn. Dirty bit/flag)

Velikost bloku Cache

- Pokud malý – mnoho bloků (každý se svojí logikou)

- Pokud velký – častější výměna (vetší p-podobnost výpadku)

často rozdělena do bloků velikosti které načteme/zapíšeme z/do DRAM jedním čtením/zápisem

Hit rate – pravděpodobnost úspěchu že dané dato je v cache, teoreticky (náhodný přístup) <<1%, prakticky až 95-99%. V případě neúspěchu (miss rate/miss penaulty) je třeba hledat dále v hiearchii.

(tj. i způsob zápisu programu/algoritmu či dat.struktur může ovlivnit hit rate a tím rychlost provedení, např dvourozměrné pole – přístup po rádkách, přístup po sloupcích)

Pro zamezení konfliktů v přístupu se často odděluje **datová cache** a **instrukční cache**.

Informace uložené v cache

- data (vlastní data) o dané velikosti bloku

- adresa (ke které adrese data náleží) (dle implementace je část asociativní a část „indexová“)

- dirty bit (příznak změny dat)

- informace o užití položky (viz dále)

Pokud potřebuji do cache nahrát novou položku musím nějakou odstranit (replacement):

- RR (random replacement) - náhodný výběr

- FIFO (first in, first outú - (nejdříve vloženou)

- LFU (least frequently used) – nejméně často používaná

- MFU (most frequently used) – nejčastěji použitá

- LDU (last recently used) – nejdéle nepoužívaná

- atp, viz např. https://en.wikipedia.org/wiki/Cache_replacement_policies

Jsou li za sebou dvě cache (L1,L2) mohou používat odlišný algoritmus.

Plně asociativní Cache

- ideální asociativní pamět, každá položka má vlastní porovnávací mechanismus – náročné

např. Adresa 32bitů, Cache o 1024 položkách, data velikosti 4B

Každý řádek cache obsahuje: 30bitů adresa (32b-2b, 2b=velikost dat), 32bitů data, Příznak platnosti 1b (je položka obsazena) + Dirty bit 1b + ?bits na nahrazovací algoritmus

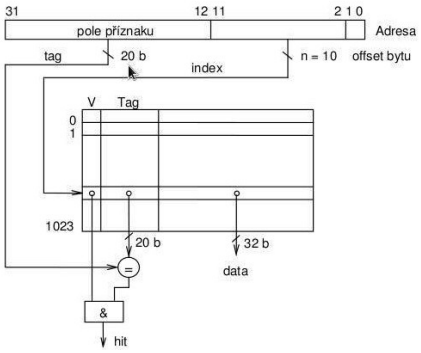
Porovnávací (asociativní část, xor fce) – je 30 bitová o musí ji obsahovat každý řádek, tj. 1024*

Cache s přímým mapováním

Spodní část adresy je přímo indexem do cache – tj. V cache nemohou být dvě adresy které mají tento index stejný.

např. Adresa 32bitů, Cache o 1024 položkách, data velikosti 4B

Každý řádek cache obsahuje: 20bitů adresa (32b-2b-10b, 2b=velikost dat, 10b=velikost indexu), 32bitů data, , Příznak platnosti 1b (je položka obsazena) + Dirty bit 1b + ?bits na nahrazovací algoritmus

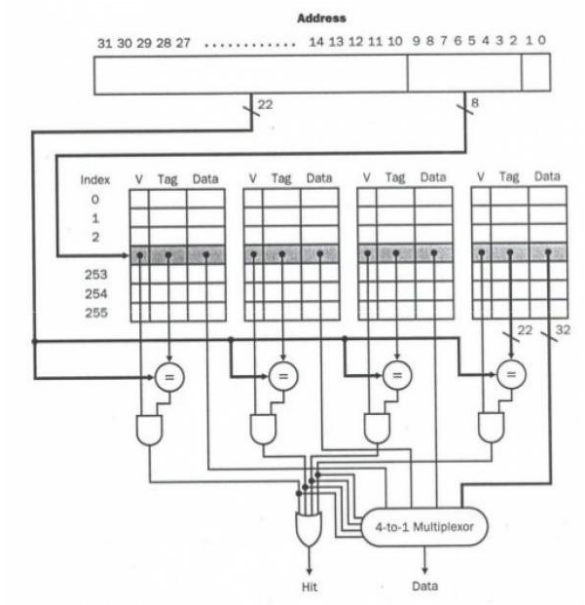


Porovnávací (asociativní část, xor fce) – je 20 bitová a je zde pouze 1*

N-cestná Cache

Spodní část adresy je indexem do cache, ale cache je vícecestná, tj. Obsahuje N bloků s daným indexem.

např. Adresa 32bitů, 4cestná cache o (celkem) 1024 položkách, data velikosti 4B



Každý řádek cache obsahuje: 22bitů adresa (32b-2b-8b, 2b=velikost dat, 8b=velikost indexu (10b-2b za čtyřcestnost)), 32bitů data, Příznak platnosti 1b (je položka obsazena) + Dirty bit 1b + ?bits na nahrazovací algoritmus

Porovnávací (asociativní část, xor fce) – je 22 bitová a je zde 4* (proto čtyřcestná)

Algorimus Pseudo-LRU

Upravený (zjednodušený) algoritmus pro určení nejdéle nepoužité položky. Není tak přesný, ale implementačně jednoduchý a pro N-položek (cest) vyžaduje 2*log2(N)-1 bitů.

Příklad pro 4 cestnou pamět – zapisujeme (na stejný index) postupně hodnoty ABCDE. Pro implementaci potřebujeme 3bity. Ty si představíme jako ukazatele v binárním stromu (0=doleva,1=doprava):

Začínáme počátečním prázdným stavem. Zapisujeme na místo kam směřují ukazatele, po zápisu ukazatelům po cestě změníme hodnotu: