第5章 メモリ・アーキテクチャ

大阪大学 大学院 情報科学研究科
今井 正治

E-mail: arch-2014@vlsilab.ics.es.osaka-u.ac.jp

講義内容

- メモリ・システムの概要
- キャッシュ・メモリ
- キャッシュの性能
- 柔軟性の高いブロック配置によるキャッシュ・ミスの削減
- 仮想記憶

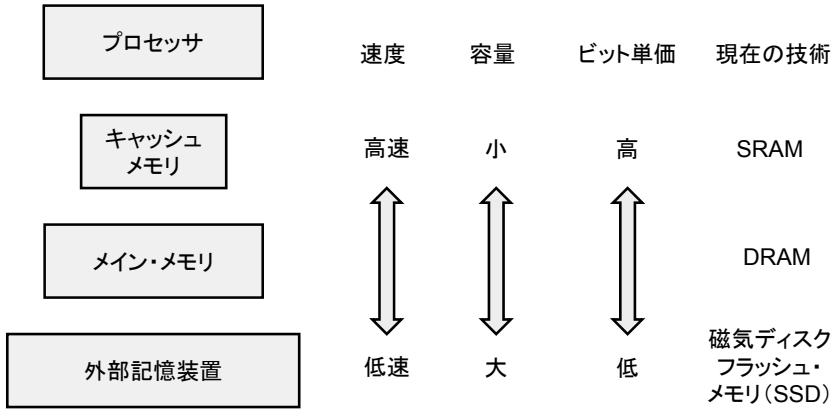
メモリ・システムの概要

- プログラマの要求
  - 高速で大容量のメモリが欲しい
- 現実
  - 高速なメモリは高価格
  - 大容量のメモリは低速
- 妥協点
  - プログラマから見て、あたかも高速で大容量のメモリがあるかのように振る舞うメモリシステム

局所性の法則（principle of locality）

- 時間的局所性（temporal locality）
  - ある項目が参照された場合、その項目がまもなく再び参照される確率が高い
- 空間的局所性（spatial locality）
  - ある項目が参照された場合、その項目の近くにある項目がまもなく参照される確率が高い
- プログラムにおける局所性
  - ループの存在 ⇒ 命令列やデータへのアクセスの時間的および空間的局所性
  - 配列への逐次的アクセス ⇒ 空間的局所性

記憶階層の基本構造



メモリのアクセス時間と価格

メモリ・テクノロジ	アクセス時間	1GBあたりの価格
SRAM	0.5 ～ 2.5 ns	2,000 ～ 5,000 USD
DRAM	50 ～ 70 ns	20 ～ 75 USD
フラッシュメモリ	25 ～ 90 ns	0.5 ～ 6 USD
磁気ディスク	5 ～ 20 ms (5,000,000 ～ 20,000,000 ns)	0.2 ～ 2 USD

メモリに関する用語(1)

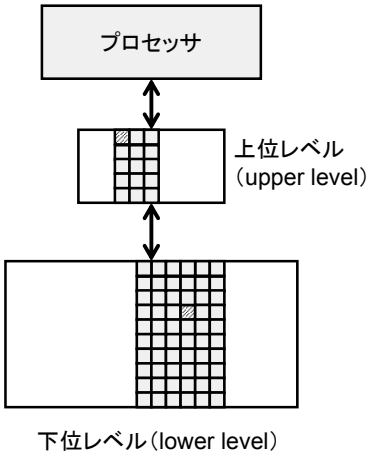
- ブロック(block), ライン(line)
  - 2つの隣接するレベル間で転送されるデータの最小単位
- ヒット(hit)
  - プロセッサから要求されたデータが上位レベルのどこかのブロックに存在する場合
- ミス(miss)
  - プロセッサから要求されたデータが上位レベルのどのブロックにも存在しない場合

メモリに関する用語(2)

- ヒット率(hit rate, hit ratio)
  - プロセッサから要求されたデータが上位レベルのどこかのブロックに存在する確率
- ミス率(miss rate, miss ratio)
  - アクセスしたデータが上位レベルで見つからない確率(= 1 – ヒット率)
- ヒット時間(hit time)
  - 記憶階層の上位レベルでのアクセスに必要な時間

メモリに関する用語(3)

- ミスペナルティ（miss penalty）
  - 上位レベルでミスした場合に、下位レベル中の求めるデータ・ブロックで上位レベルの対応するブロックを置き換え、そのブロックをプロセッサに取り込むのに要する時間



講義内容

- メモリ・システムの概要
- キャッシュ・メモリ
- キャッシュの性能
- 柔軟性の高いブロック配置によるキャッシュ・ミスの削減
- 仮想記憶

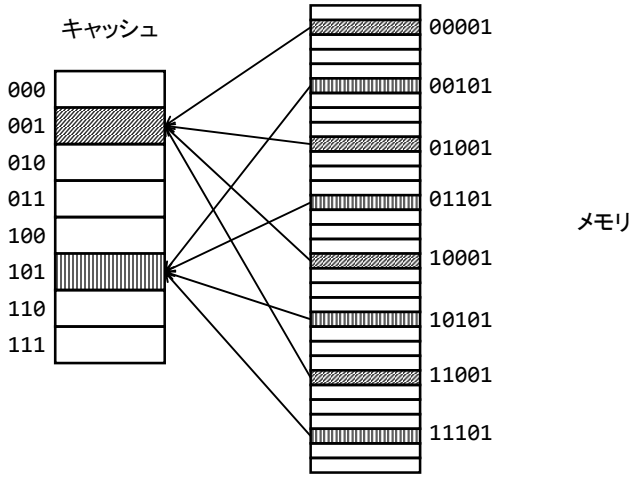
キャッシュ・メモリ（cache memory）

- データ項目の一部を高速なキャッシュ（SRAM）に格納する
- 解決すべき課題
  - データ項目がキャッシュ内に存在するかどうか?
  - データ項目がキャッシュ内に存在する場合、それがどの位置にあるかをどのようにして知るか？
- 最も簡単なマッピング方法
  - ダイレクト・マッピング（direct mapping）

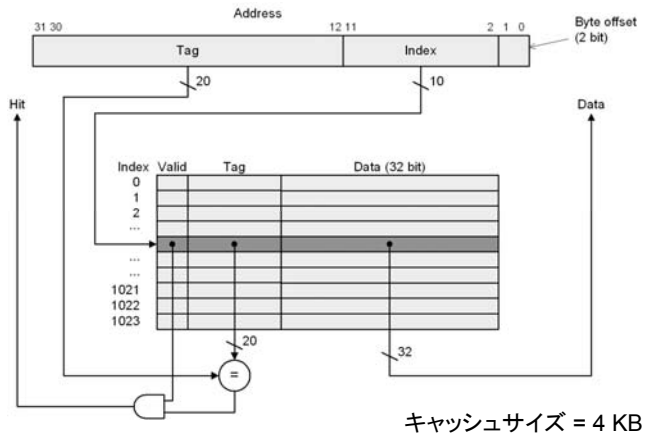
ダイレクト・マッピング

- キャッシュ中のデータ項目の位置（index）
  =（ブロックのアドレス）mod（ブロック数）
- ブロック数が2のベキの場合、ブロックのアドレスの下位のビットがそのままデータ項目の位置になるので計算が簡単
- キャッシュに追加する項目
  - 有効ビット（valid bit）: キャッシュ中のデータ項目が有効かどうかを表す情報
  - タグ（tag）: キャッシュ中のデータ項目のアドレス情報（マッピングに使用しなかったアドレスの上位のビット）

ダイレクト・マッピング



アドレスとキャッシュ・インデックスの対応



キャッシュサイズ = 4 KB

キャッシュへ・アクセスの例

参照先の10進アドレス	参照先の2進アドレス	ヒット/ミス	割当てられるキャッシュ・ブロック
22	10110	Miss	10110 mod 8 = 110
26	11010	Miss	11010 mod 8 = 010
22	10110	Hit	10110 mod 8 = 110
26	11010	Hit	11010 mod 8 = 010
16	10000	Miss	10000 mod 8 = 000
3	00011	Miss	00011 mod 8 = 011
16	10000	Hit	10000 mod 8 = 000
18	10010	Miss	10010 mod 8 = 010
16	10000	Hit	10000 mod 8 = 000

キャッシュの内容の推移(1)

□ 初期状態(電源投入後)

インデックス	有効	タグ	データ
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

キャッシュの内容の推移(2)

□ アドレス 10110 のミスを処理した後

インデックス	有効	タグ	データ
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	メモリ(10110)
111	N		

キャッシュの内容の推移(3)

□ アドレス 11010 のミスを処理した後

インデックス	有効	タグ	データ
000	N		
001	N		
010	Y	11	メモリ(11010)
011	N		
100	N		
101	N		
110	Y	10	メモリ(10110)
111	N		

キャッシュの内容の推移(4)

□ アドレス 10000 のミスを処理した後

インデックス	有効	タグ	データ
000	Y	10	メモリ(10000)
001	N		
010	Y	11	メモリ(11010)
011	N		
100	N		
101	N		
110	Y	10	メモリ(10110)
111	N		

キャッシュの内容の推移(5)

□ アドレス 00011 のミスを処理した後

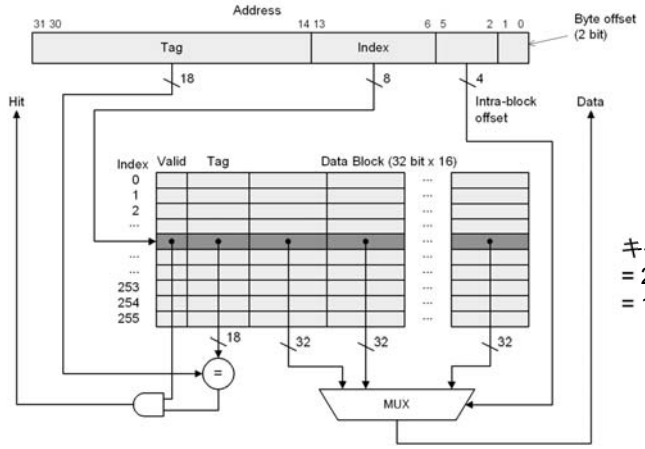
インデックス	有効	タグ	データ
000	Y	10	メモリ(10000)
001	N		
010	Y	11	メモリ(11010)
011	Y	00	メモリ(00011)
100	N		
101	N		
110	Y	10	メモリ(10110)
111	N		

キャッシュの内容の推移(6)

□ アドレス 10010 のミスを処理した後

インデックス	有効	タグ	データ
000	Y	10	メモリ(10000)
001	N		
010	Y	10	メモリ(10010)
011	Y	00	メモリ(00011)
100	N		
101	N		
110	Y	10	メモリ(10110)
111	N		

ブロックサイズが16語のキャッシュ



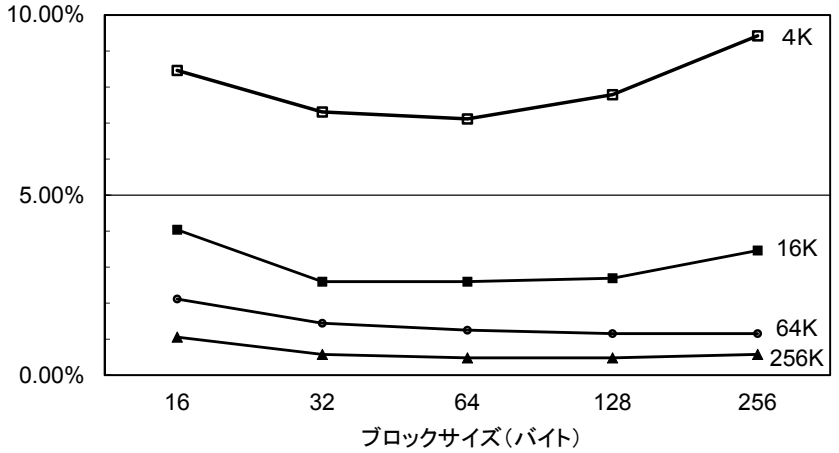
キャッシュサイズ
= 256 x 16 x 4
= 16 KB

キャッシュのビット数

□ 仮定
- 32ビットのバイトアドレス
- ダイレクト・マップ方式
- キャッシュ総量 2^n ブロック(インデックス n ビット)
- ブロックサイズ 2^m 語(2^{m+2} バイト)

□ 計算方法
- タグフィールド長 = $32-(n+m+2)$
- キャッシュの総ビット数
  $2^n \times$ (ブロックサイズ+タグ長+有効フィールド長)
  $= 2^n \times (2^m \times 32 + (32-n-m-2)+1)$
  $= 2^n \times (2^m \times 32 + 31 - n - m)$

ブロックサイズとミス率の関係



ブロックサイズとミス率の関係

- ブロックが大きいほど，ミス率が低下する（空間的局所性）
- 通常は，ブロックサイズを大きくするとミス率が低下する
- ただし，キャッシュ容量に対するブロックの相対的サイズを非常に大きくすると，ミス率は逆に高くなる（例：キャッシュサイズ4K，ブロックサイズ256バイトの場合）
- ブロックサイズが大きすぎると，空間的局所性が低下する（ブロック数が少なくなるため）
- ブロックサイズが大きいと，ミス・ペナルティも増大
- 最適なブロックサイズを選択すべき

命令キャッシュでのキャッシュ・ミスの取り扱い

1. 元のPCの値（PC-4）をメモリに送る
2. 主記憶から読出しを行うよう指示し、その完了を待つ
3. キャッシュの該当するブロックに書込みを行う
   1. 主記憶から読み出したデータをキャッシュのデータ部に格納
   2. アドレスの上位ビットをタグ・フィールドに格納する
   3. 有効ビットをオンに設定する
4. 命令実行を最初のステップから再開する（命令はキャッシュ中にある）

書き込みの取り扱い

- データの書き込みを行う場合、データをキャッシュにだけ書き込むと、キャッシュの内容と主記憶の内容が異なってしまう
  ⇒ キャッシュの一貫性（consistency）の喪失
- 一貫性を保つ方法
  - ライト・スルー（write through）
  - ライト・バック（write back）

ライト・スルー

- データの書き込みが発生した場合、データをキャッシュと主記憶の両方に書き込む
- 利点
  - 制御が簡単
- 問題点
  - 性能の低下（データの書き込みが発生するたびに主記憶へのデータの書き込みが行われる）
- 性能低下への対策
  - ライト・バッファ（write buffer）の採用
  - 書き込むデータをライト・バッファに書き込んだら、主記憶へのデータ書込みの完了を待たず演算処理を継続

ライト・バック

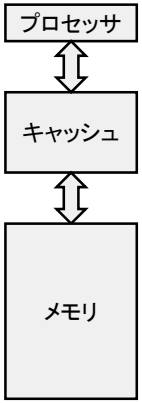
- 別称: コピー・バック(copy back)
- 処理方法
  - 書き込みが発生したとき、新しい値はキャッシュ内の対応するブロックにのみ書き込む
  - 主記憶の内容が更新されるのは、キャッシュ内のブロックが置き換え対象になった場合のみ
- 利点
  - 高性能
- 問題点
  - 制御が複雑

記憶システムの設計

- 仮定
  - アドレスの送出に必要な時間
    1 メモリ・バス・サイクル
  - DRAMの1語当たりのアクセス時間
    15 メモリ・バス・サイクル
  - 1語当たりのデータ転送に必要な時間
    1 メモリ・バス・サイクル
  - キャッシュの1ブロックは4語から構成される

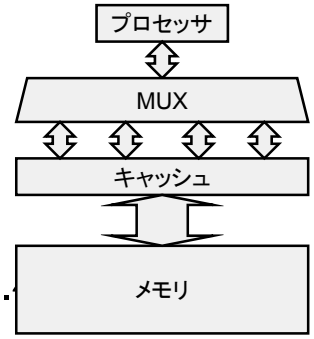
1語幅のメモリ構成

- ミス・ペナルティ
  $1 + 4 \times 15 + 4 \times 1 = 65$
- 1メモリ・バス・サイクル
  当たりの転送バイト数
  (バンド幅: band width)
  $(4 \times 4)/65 = 0.25$



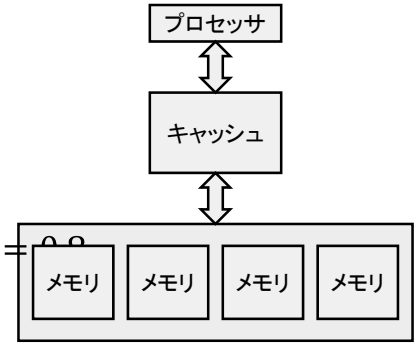
データバス幅を広げたメモリ構成

- メモリのバス幅: 32×2
- ミス・ペナルティ
  $1 + 2 \times 15 + 2 \times 1 = 33$
- 1メモリ・バス・サイクル
  当たりの転送バイト数
  $(4 \times 4)/33 = 0.$

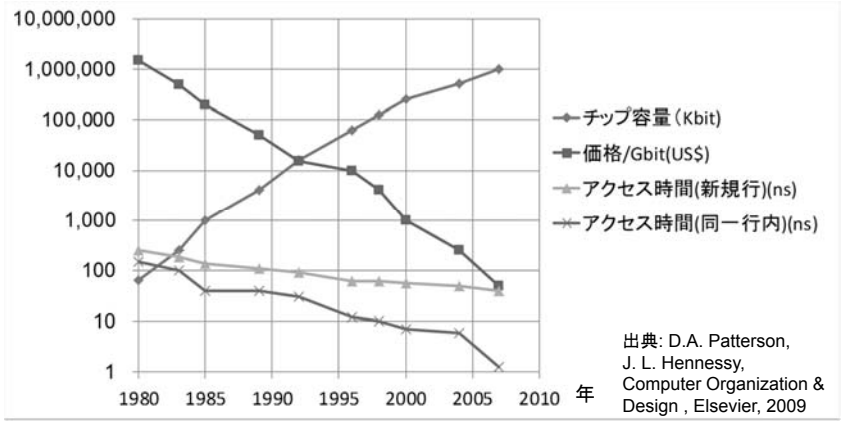


インタリーブ方式のメモリ構成

- 1ブロック(4語)あたりのミス・ペナルティ
  $1 + 15 + 4 \times 1 = 20$
- 1メモリ・バス・サイクル当たりの転送バイト数
  $(4 \times 4)/20 = 0.8$



DRAMの進歩の様子



出典: D.A. Patterson, J. L. Hennessy, Computer Organization & Design , Elsevier, 2009

講義内容

- メモリ・システムの概要
- キャッシュ・メモリ
- キャッシュの性能
- 柔軟性の高いブロック配置によるキャッシュ・ミスの削減
- 仮想記憶

キャッシュの性能(1)

- CPU時間 =
  (CPU実行クロック数＋メモリ・ストール・クロック数)
  ×クロック・サイクル時間
- メモリ・ストール・クロック数 =
  読出しストール・クロック数
  ＋書き込みストール・クロック数
- 読出しストール・クロック数 =
  プログラム当たりの読出し回数
  ×読出しミス率
  ×読出しミス・ペナルティ

ライトスルー方式での書き込みストール・クロック数

- 書き込みストール・クロック数 =
  プログラム当たりの書き込み回数
  ×書き込みミス率
  ×書き込みミス・ペナルティ
  ＋書き込みバッファ・ストール
- 書き込みバッファ・ストールは単純には表現できない
  - 適切な深さ（容量）のライト・バッファとプログラムの平均書き込み頻度を超える性能のメモリシステムがあれば、無視できる

キャッシュの性能（2）

- データ・メモリ・ストール・クロック数 =
  プログラム当たりのメモリアクセス回数
  ×ミス率
  ×ミス・ペナルティ
- 命令メモリ・ストール・クロック数 =
  プログラム当たりの命令アクセス回数
  ×1命令アクセス当たりのミス回数
  ×ミス・ペナルティ

キャッシュの性能の計算例（1）

- 仮定
  - 命令のキャッシュ・ミス率 2%
  - データのキャッシュ・ミス率 4%
  - CPI (Cycle Per Instruction) 2
    （メモリ・ストールを除く）
  - ミス・ペナルティ 100 クロック・サイクル
  - 命令数 *N*
  - ロード命令およびストア命令の割合 36%

キャッシュの性能の計算例（2）

- 命令ミスのクロック数
  - $N \times 2\% \times 100 = 2.00\ N$
- データミスのクロック数
  - $N \times 36\% \times 4\% \times 100 = 1.44\ N$
- メモリ・ストールの合計クロック数
  - 2.00 + 1.44 = 3.44
- メモリのストールを考慮したCPI
  - 2 + 3.44 = 5.44

キャッシュの性能の計算例(3)

- 完全な(ミスを起こさない)キャッシュを持つCPUのCPI
  - 2.00
- 完全なキャッシュを持つCPUとの性能比
  - 2.00 / 5.44 = 0.368

平均メモリ・アクセス時間

- AMAT (Average Memory Access Time)
  = ヒットした場合のアクセス時間
  + ミス率 × ミス・ペナルティ
- 例
  - クロックサイクル時間 1ns
  - ミス・ペナルティ 20 クロック・サイクル(= 20 ns)
  - 命令当たりのミス率 5% (0.05)
  - キャッシュアクセス時間 1クロックサイクル (= 1 ns)
  - AMAT = 1 + 0.05 × 20 = 2 ns

講義内容

- メモリ・システムの概要
- キャッシュ・メモリ
- キャッシュの性能
- 柔軟性の高いブロック配置によるキャッシュ・ミスの削減
- 仮想記憶

ダイレクト・マップ方式とフル・アソシアティブ方式

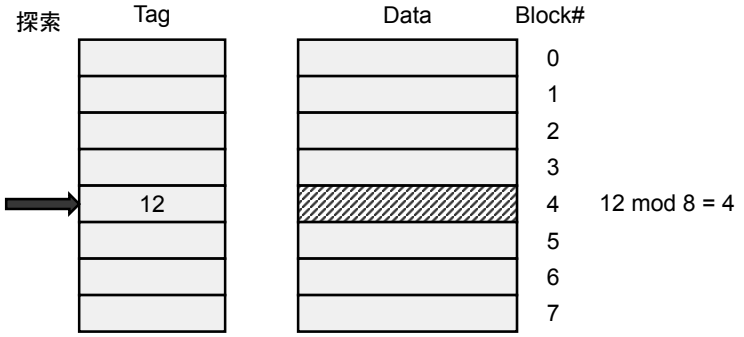
- ダイレクト・マップ方式の問題点
  - メモリ・ブロックを単一のロケーションに配置
  - 構造は簡単であるが、柔軟性に乏しい
- フル・アソシアティブ(Full Associative)方式
  - メモリ・ブロックはキャッシュ内のどのロケーションにでも配置可能
  - ヒット/ミスの判定を行うためには、キャッシュ内の全てのエントリを探索するする必要があり、ハードウェアが複雑になる

セット・アソシアティブ（Set Associative）方式

- キャッシュ内で、各ブロックを配置可能な場所がある決まった数（$n > 1$）だけ存在する
- n ウェイ（way）・セット・アソシアティブ・キャッシュ
- キャッシュは複数のセットから構成される
- 各セットには、n 個のブロックから構成される
- メモリ中の各ブロックは、インデックス・フィールドが示すキャッシュ中の特定のセットに対応する
- 1 ウェイ・セット・アソシアティブ方式
  = ダイレクト・マップ方式
- 連想度（associativity）: 1セット中のブロック数（=ウェイ数）

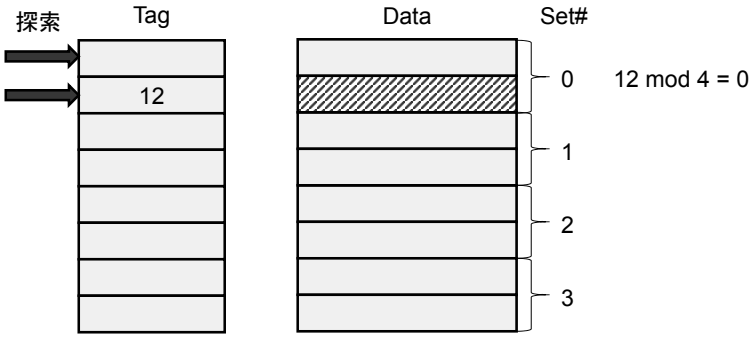
データの配置方法の比較（1）ダイレクト・マップ方式

- ブロックの位置 =
  ブロック番号 mod キャッシュ中のブロック数



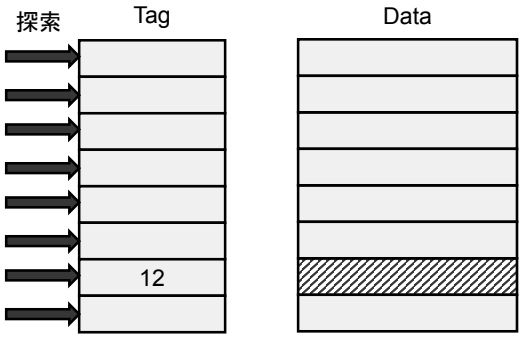
データの配置方法の比較（2）セット・アソシアティブ方式（2 way）

- ブロックの位置 =
  ブロック番号 mod キャッシュ中のセット数



データの配置方法の比較（3）フル・アソシアティブ方式

- ブロックの位置 =
  ブロック番号 mod キャッシュ中のセット数



ウェイ数とキャッシュの構成の関係

- 1 way
  （ダイレクトマップ）

Block#	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

- 2 way

Set#	Tag	Data	Tag	Data
0				
1				
2				
3				

ウェイ数とキャッシュの構成の関係

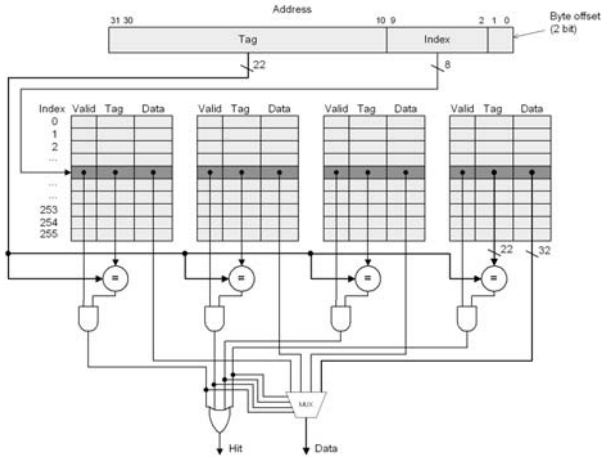
- 4 way

Set#	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

- 8 way (Full Associative)

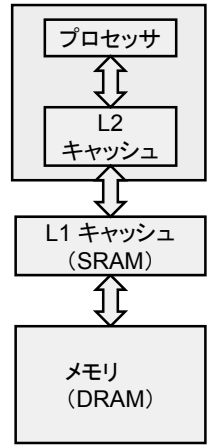
Tag	Data	Tag	Data	Tag	Data	…	Tag	Data

4ウェイ・セット・アソシアティブ方式のキャッシュの実装方法



マルチレベル・キャッシュ

- 複数のレベルのキャッシュを用いてミス・ペナルティを減少させる方式
- L2キャッシュ
  - メイン・チップに内蔵
  - 小容量・高速
- L1キャッシュ
  - メインチップとメモリの中間



マルチレベル・キャッシュの性能評価例(1)

- 仮定
  - 基本CPI: 1.0
  - クロック周波数: 4 GHz (クロック周期 0.25 ns)
  - 主記憶のアクセス時間: 100 ns (含ミスペナルティ)
  - L1キャッシュのミス率: 2%
  - L2キャッシュのアクセス時間: 5 ns (miss/hit)
  - L2キャッシュの追加により, 主記憶へのミス率が0.5%に減少

マルチレベル・キャッシュの性能評価例(2)

- 計算例
  - 主記憶に対するミス・ペナルティ
    100 ns ÷ 0.25 ns = 400 クロックサイクル
  - キャッシュが1レベルの場合の実効CPI
    基本CPI＋命令当たりのメモリ・ストール・サイクル数
    = 1.0 ＋ 2% × 400
    = 9.0
  - L2キャッシュに対するミス・ペナルティ
    5 ns ÷ 0.25 ns = 20 クロックサイクル

マルチレベル・キャッシュの性能評価例(3)

- 計算例(続き)
  - 合計CPI
    = 基本 CPI
    ＋ 命令当たりの平均ストールサイクル数(L1)
    ＋ 命令当たりの平均ストールサイクル数(L2)
    = 1.0＋2%×20＋0.5%×400
    = 1.0＋0.4＋2.0
    = 3.4
  - 性能向上率
    = 9.0 / 3.4 = 2.6 倍

キャッシュ・ミスが生じた場合に置き換えるブロックを選択する方法

- キャッシュの方式による違い
  - ダイレクト・マッピング: 一意に決まる
  - セット・アソシアティブおよびフル・アソシアティブ:
    複数の候補がある
- LRU (Least Recently Used)
  - 最も長い間使用されていなかったブロックの中から選択する
- ランダム
  - 候補の中からランダムに選択する

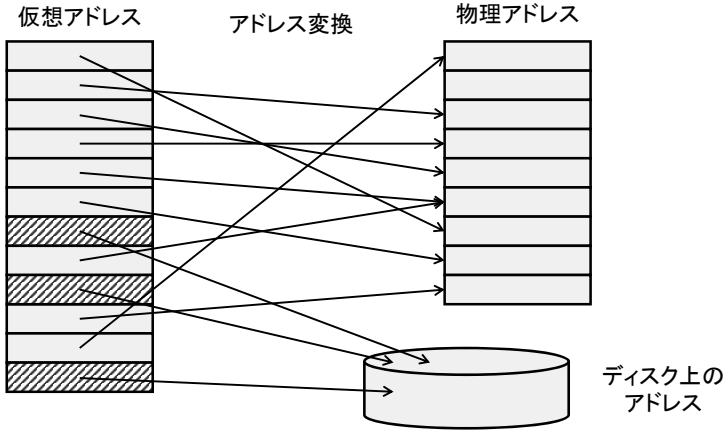
講義内容

- メモリ・システムの概要
- キャッシュ・メモリ
- キャッシュの性能
- 柔軟性の高いブロック配置によるキャッシュ・ミスの削減
- 仮想記憶

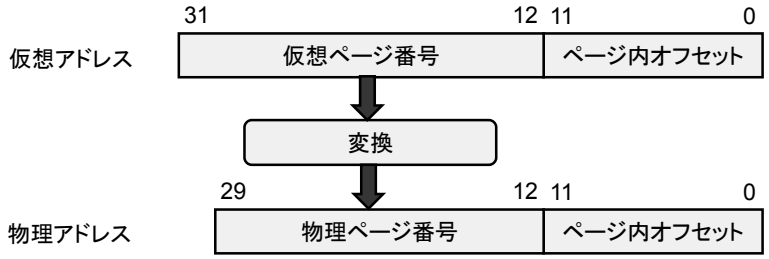
仮想記憶（virtual memory）

- 仮想記憶の開発目的
  - 複数のプログラムの間でのメモリの効率的共有
  - 主記憶容量の限界に対処するプログラミング上の負荷の軽減
    （アドレス空間を小片（オーバーレイ）に分割し，小片をユーザの管理下でロード/アンロードしていた）
- 仮想記憶の機能
  - 各プログラムのアドレス空間を物理アドレスに変換
  - 他のプログラムのアドレス空間の保護

仮想記憶の仕組み



仮想アドレスと物理アドレスとの対応付け



- 仮想アドレス空間: 32 bit ⇒ 4 GB
- ページサイズ: 12 bit（4 KB）
- 物理ページ番号: 18 bit
- 主記憶の最大容量: 30 bit ⇒ 1 GB

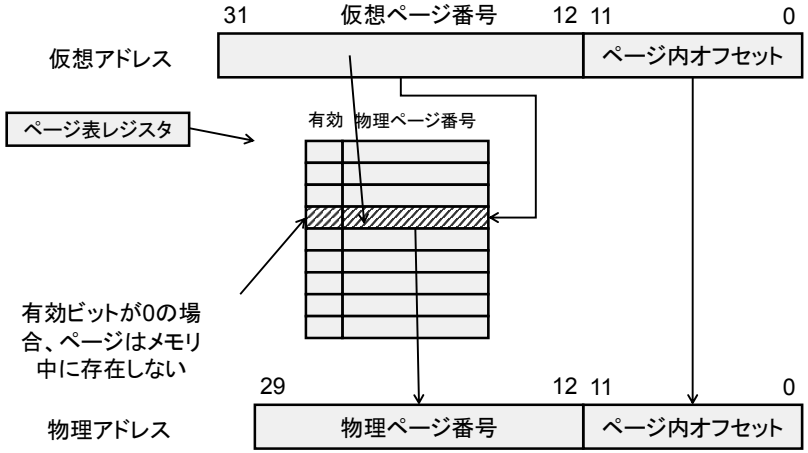
仮想記憶システムの設計上の注意点(1)

- ページのミス(ページ・フォールト: page fault)が生じた場合のペナルティが大きい(数100万サイクル)
- 長いアクセス時間を補えるよう、ページサイズを十分大きく取る
  - 典型的なページサイズは4KB～16KB
  - 最新のサーバシステムのページサイズは 32KB～64KB
- ページフォルトの発生を少なくする構成
  - フル・アソシアティブ方式の採用

仮想記憶システムの設計上の注意点(2)

- ページフォルトは、ソフトウェアで対応
  - ディスクアクセスに要する時間に比べて、ソフトウェアのオーバヘッドは少ない
  - ページフォルトの発生を少なくするアルゴリズムを採用
- ライト・バック方式の採用
  - 書き込みに長い時間がかかるので、ライト・スルー方式を採用するメリットはない
  - ディスクへのアクセス回数を減らす方が効果的

ページ表の仕組み



アドレス変換バッファ

- TLB: Translation-Lookaside Buffer
  - タグ
    - 仮想アドレスを保持
  - ダーティ(dirty)
    - ライト・バック方式で、ページ内に書き込みがあったかどうかの情報を保持
  - 参照(reference)
    - 一定期間内にアクセスがあったかどうかの情報を保持 ページ置き換えの際に使用(疑似LRU)

TLBの機能