キャッシュとキャッシュ技術

定義

アクセス要求を発行する機構と，その供給に応える機構との中間に位置し，すべての要求を検知して処理するよう構築される．

キャッシュは選択されたデータの局所的なコピーを保持し，可能な場合にはアクセス要求にこたえる．

通常のメモリ機構より高速に動作するよう設計されている

メモリアクセス時間の短縮など，性能向上を目指す．

メモリ	アクセス時間	1G当たりのコスト
SRAM	0.5～2.5nS	2000～5000ドル
DRAM	50～70nS	20～75ドル
磁気ディスク	5～20mS	0.2～2ドル

- 時間的局所性と空間的局所性

- キャッシュの特徴
  - 小容量

    メインメモリの10%程度の小容量
  - 常時動作

    要求されたデータがキャッシュで利用可能か

    可能でないなら，メインメモリからのコピーを取り出したり，どのデータをキャッシュ上に保持するか決定する機構
  - 透過性

    要求側から見えるインターフェイスは，メインメモリに示すインターフェイスと同一
  - 自動性

    どのデータを保持するかなどの命令はない

- キャッシュ技術の重要性

  情報を検索するほぼすべてのハードウエアやソフトウエアシステムにおいて利用される，基本的な最適化技術

  キャッシュ内に保持されたデータが特定の形式や，サイズ制限されない

  - 小規模データ（バイトやワードメモリ）
  - 中規模データ（メモリのセグメントやページ）
  - 大規模データ（プログラム全体）
  - 包括的なデータ（ファイルやディスクブロック）
  - アプリケーションに特化したデータ

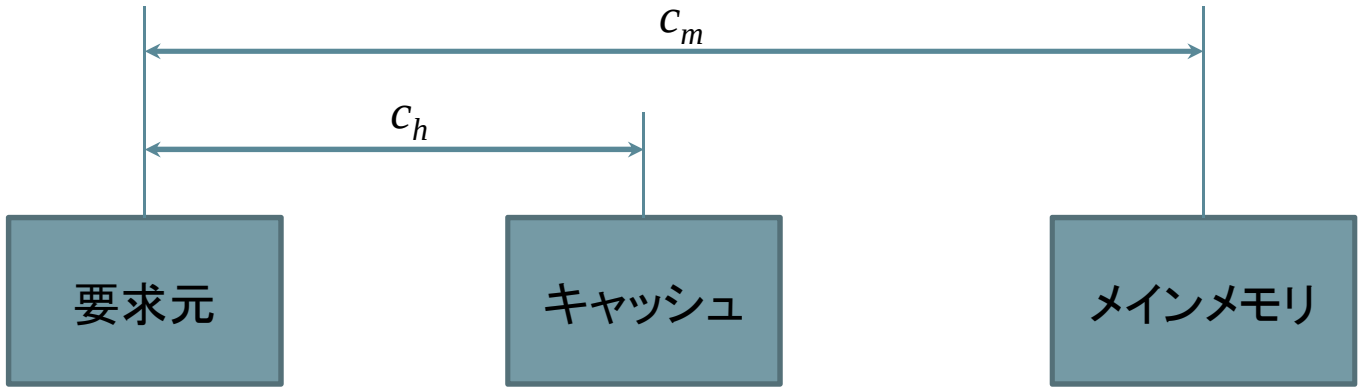
    （Webページやワープロ文書，データベース登録データ）
  - 文書データ（電子メールなど）

- キャッシュにおける用語
  - キャッシュヒット

    メインメモリへのアクセスを必要とせず，要求がキャッシュによって満足されること
  - キャッシュミス

    キャッシュによっては，要求が満足されないこと

- 最善，最悪の場合のキャッシュ性能
  - ヒットした場合のコスト c_h
  - ミスヒット時のコストc_m



- N個の連続したアクセス列についての，最良，最悪の振舞い
  - すべてのアクセスがあらたなデータを参照する場合：最悪時
  - キャッシュによる性能の改善はない → 最悪時のコスト c_{worst}

    $c_{worst}=Nc_m$

    アクセスごとの平均コスト＝ c_m
  - 連続するすべてのアクセスが，同じデータを指す場合
  - キャッシュによる性能の改善は最良 → 最善時のコスト c_{best}

    $c_{worst}=c_m+(N-1)\,c_h$

    アクセスごとの平均コスト $\frac{c_m+(N-1)c_h}{N}=\frac{c_m}{N}-\frac{c_h}{N}+c_h$

    $N \to \infty$ ：平均コスト $\to c_h$
- キャッシュによる性能は，キャッシュが存在しない場合に比べ悪くはならない

- 典型的な連続アドレスにおけるキャッシュ性能
  - ヒット率 $=\dfrac{\text{ヒットしたアクセス数}}{\text{全アクセス数}}$
  - ミス率＝1－ヒット率
  - データ記憶にアクセスするコスト
  - コスト＝$rc_h + (1-r)c_m$　r:ヒット率
  - キャッシュ性能の改善：ヒット率の向上
    ヒット時のコストの低減

- キャッシュ置き換えポリシー

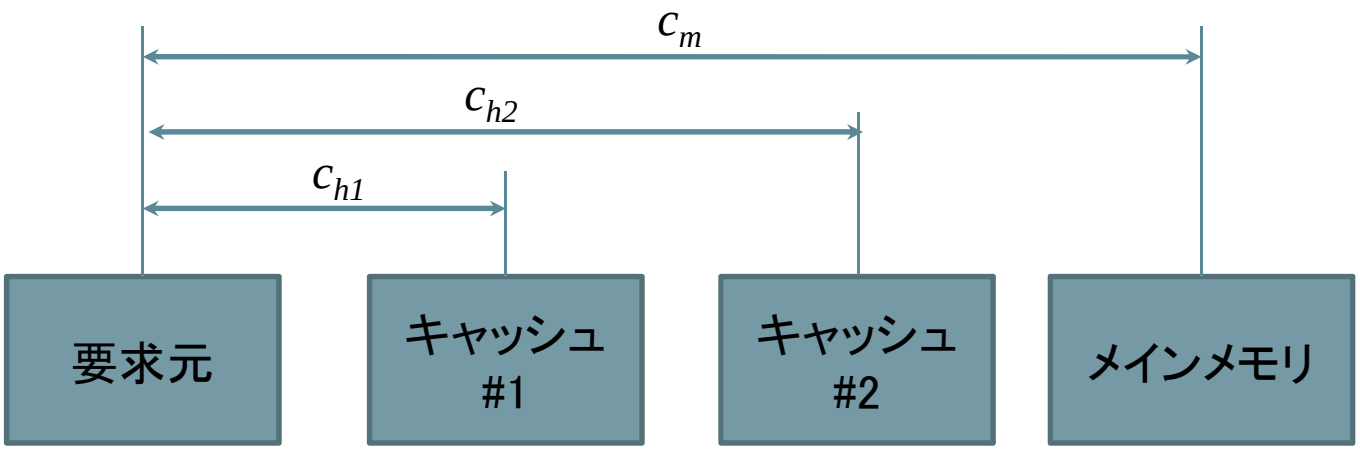
  新たなデータを無視するのか，新たなデータのための場所を確保するために，どの古いデータをキャッシュ上から消去するのか

- LRU（Least Recently Used)置き換え
  - 最も長い期間参照されなかったデータを置き換える
  - キャッシュメカニズムは現在キャッシュ上にあるデータ項目のリストを保持
  - データの参照後，リストの最前部に移動
  - データの置き換えはリストの最後部から

- 多重レベルキャッシュ階層

$$\text{コスト} = r_1 c_{h1} + r_2 c_{h2} + (1 - r_1 - r_2) c_m$$

r_1, r_2:ヒット率



- 先読みキャッシュ
- システム起動時：

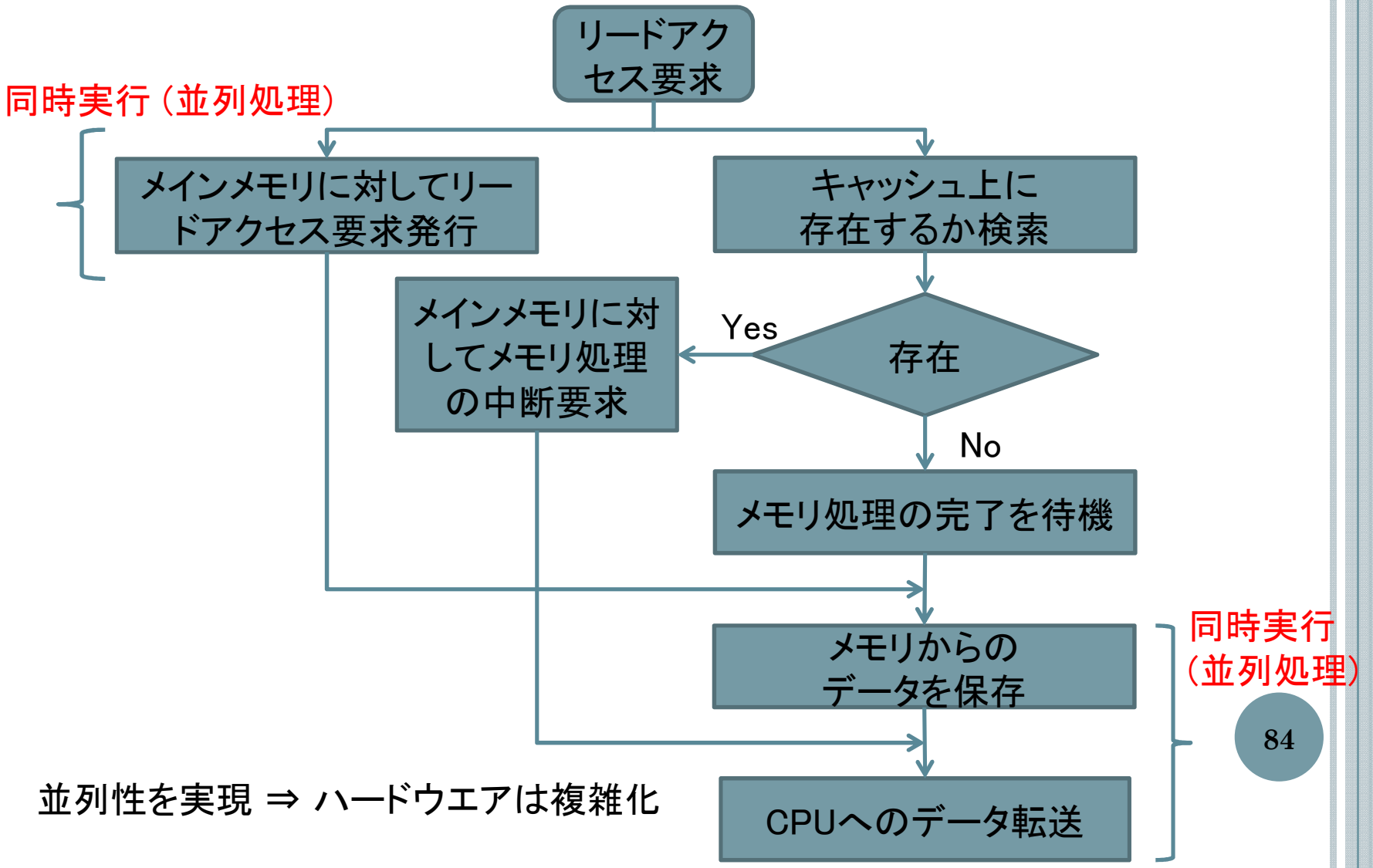
  キャッシュがメインメモリよりデータを読み出すため初期ヒット率は極端に低下
- キャッシュの先読み(pre-load)により，起動時の負荷を低減
- 関連するデータを先読み(pre-fech)

  プロセッサが1バイトアクセスする際，64バイトプリフェッチ　2バイト目からはキャッシュがヒット

- メモリシステムにおけるキャッシュ
- メモリ：高価で低速

  キャッシュ：高速メモリの高いコストをかけずに性能改善

物理メモリキャッシュ



並列性を実現 ⇒ ハードウエアは複雑化

- メモリキャッシュの実現

  キャッシュのエントリ

    メモリアドレスとそのアドレスで示されるバイト列

  各エントリごとに完全なアドレスを保持することは非効率

  必要となる空間の容量削減のための技術

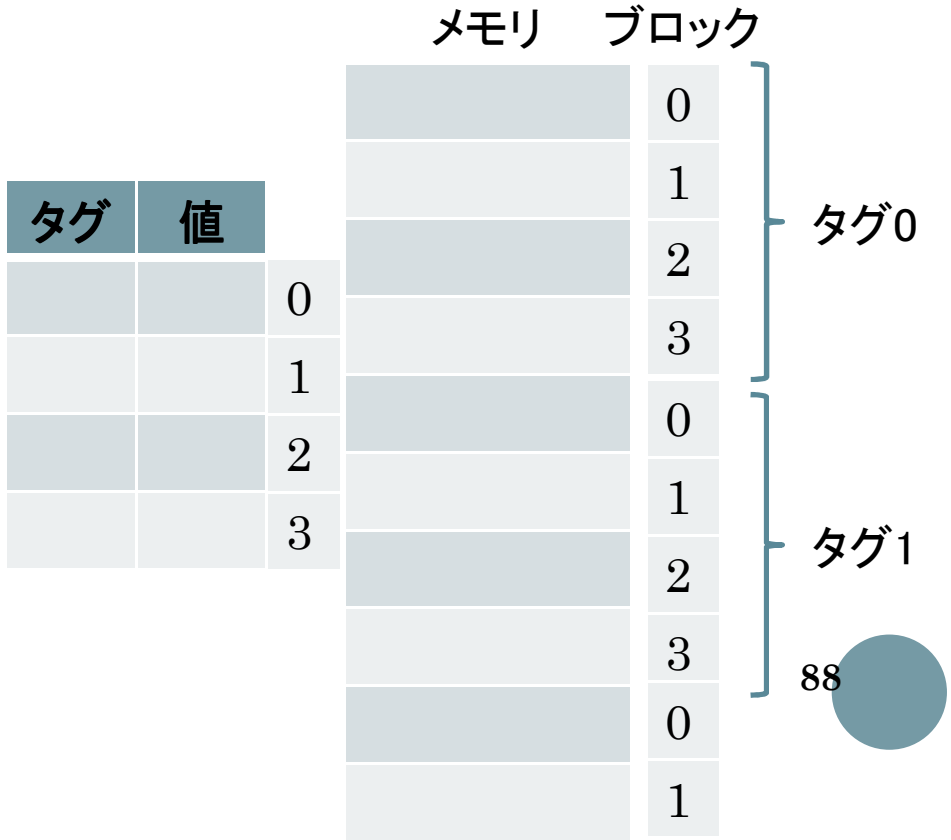
    - ダイレクトマッピング
    - セットアソシアティブ

- ダイレクトマッピングキャッシュ
  - 2つのアドレスはキャッシュ内の1つの空きスロットを奪い合う
  - A1への参照は，キャッシュ内のA1の値を読み出し，A2への参照はA2の値を読み出す
  - 交互に参照すると，すべての参照はキャッシュミス
- セットアソシアティブキャッシュ
  - A1が2つのキャッシュ内の1つに置かれ，A2はもう一方に格納することができる
  - 交互の参照でも，すべてキャッシュはヒットする
- 並列度が増すと，性能は向上

- セットアソシアティブキャッシュ
  - 複数のキャッシュを管理
  - 同時にそれらすべてを検索できるハードウエア
  - 複数のキャッシュを扱うため，同じ番号を持つブロックを1つ以上格納可能

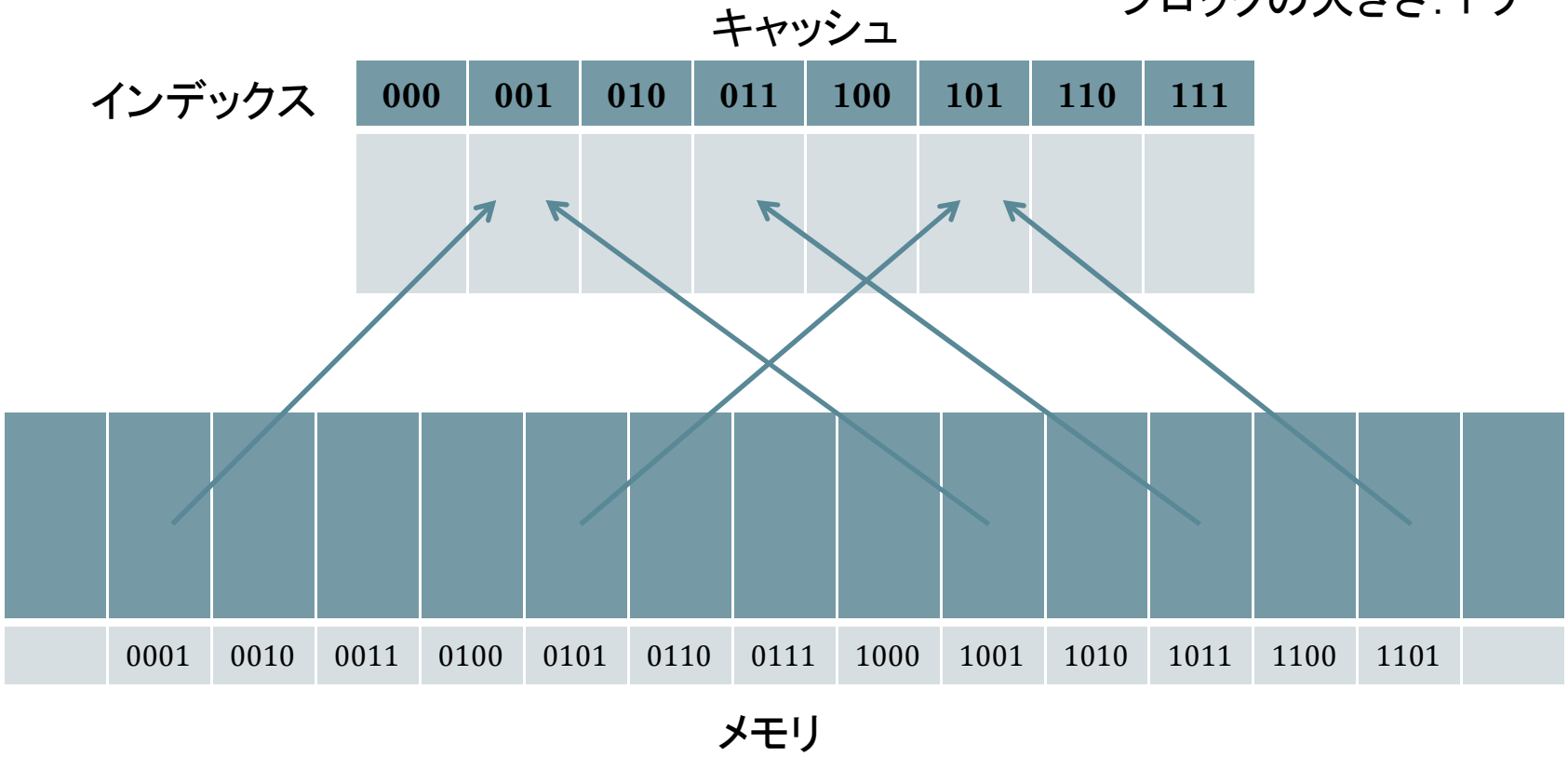
- ダイレクトマッピング方式のキャッシュ
  - キャッシュはバイトアドレッシング
  - メモリとキャッシュを同一のサイズのブロック群に分割
  - ブロックごとのメモリの取り扱い

	メモリ			ブロック
0	1	2	3	0
4	5	6	7	1
8	9	10	11	2
12	13	14	15	3



ダイレクトマッピング方式

ブロック数：8
ブロックの大きさ：1ワード



0 0 0 1

タグ　キャッシュ中のブロック番号

- ダイレクトマッピング方式のキャッシュの動作例

参照先の10進アドレス	参照先の2進アドレス	ヒット/ミスの別	割り当てられているキャッシュ・ブロック
22	10110	ミス	110
26	11010	ミス	010
22	10110	ヒット	110
26	11010	ヒット	010
16	10000	ミス	000
3	00011	ミス	011
16	10000	ヒット	000
18	10010	ミス	010
16	10000	ヒット	000

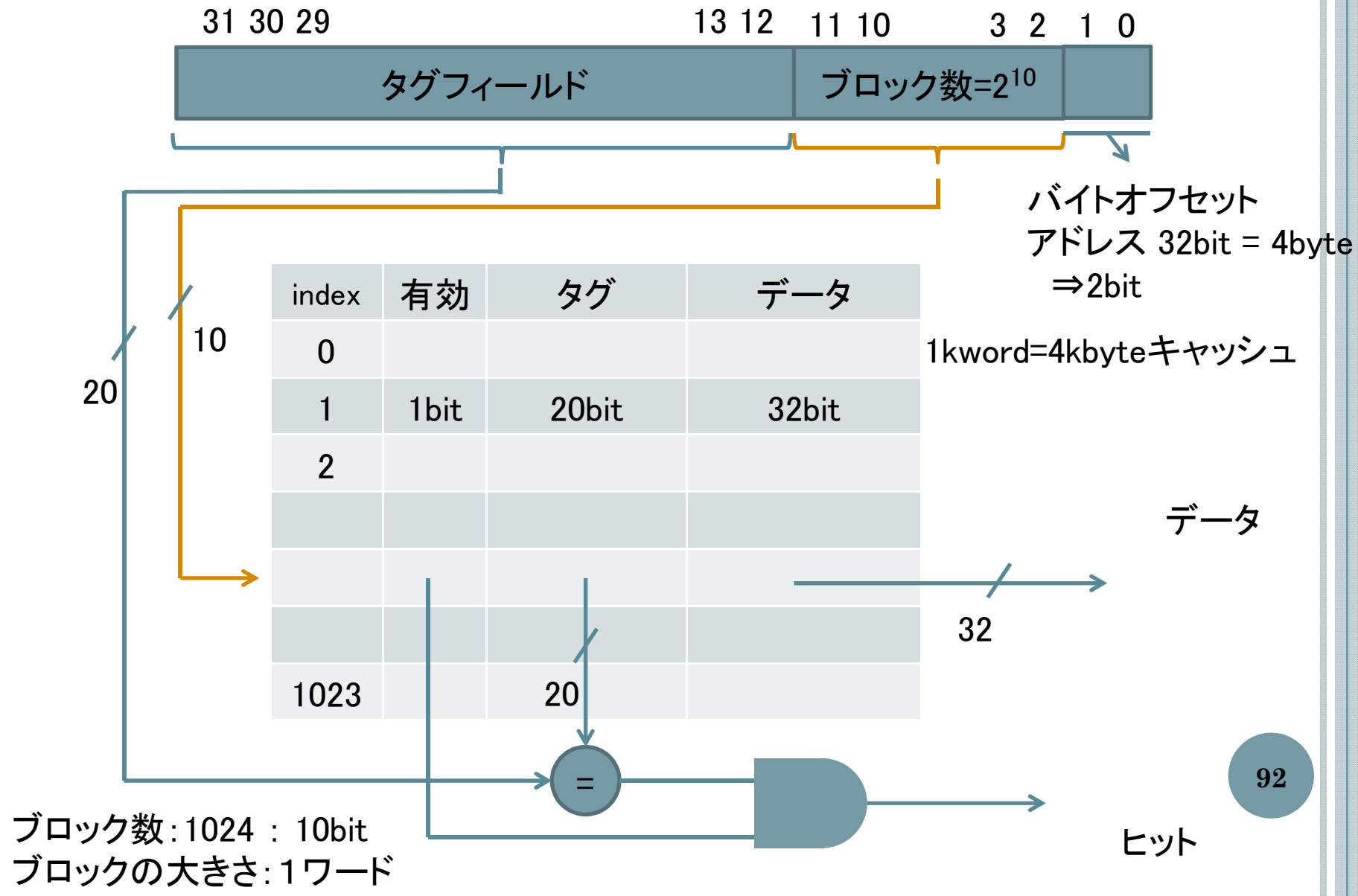
インデックス	有効	タグ	データ
000	N		
001	N		
010	N		
011	N		
100	N		
110	N		
111	N		

電源投入直後

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

アドレス 10110 のミスを処理した後

アドレスとキャッシュインデックスの関係



キャッシュ容量：2^nブロック ⇒ キャッシュインデックス：n bit

ブロックサイズ： 2^m ワード（= 2^{m+2}バイト）

1ブロック＝2^m ワード

ブロック番号	有効	タグ	#0ワード	#1ワード	…	#(2^m-1)ワード
0	1	32-(n+m+2) bit	32 bit	32 bit	32 bit	32 bit
1						
…						
2^n−1						

2^n ×（有効フィールド長 ＋ タグ長 ＋ ブロックサイズ）
＝ 2^n ×（ 1＋(32−(n＋m＋2) ＋2^m ×32）

16kバイトのデータを保持するダイレクトマップ方式のキャッシュに必要なビット数．ブロックサイズは4word，アドレスは32bitとする

16kバイトのデータを保持するダイレクトマップ方式のキャッシュに必要なビット数．ブロックサイズは4word，アドレスは32bitとする

1word=4byte，16kbyte=4kword，ブロックサイズが4word

キャッシュのブロック数=1k=2^{10}

	有効	タグ	#0word	#1word	#2word	#3word
0	1	18	32	32	32	32
1						
1023						

$1024 \times (1 + 18 + 4 \times 32) = 1024 \times 147 = 147\text{kbit}$

ブロックサイズが16バイトの64個のブロックからなるキャッシュがある. バイトアドレスが1200番地のブロック番号はいくらか

ブロックサイズが16バイトの64個のブロックからなるキャッシュがある．バイトアドレスが1203番地のブロック番号はいくらか

ブロックアドレスは $\left\lfloor \frac{1203}{16} \right\rfloor$=75

このブロックアドレスに対するキャッシュブロック番号は75を64で割った余りの11

ちなみにこのブロック番号75のブロックには，1200番地から1215番地のバイトアドレスに対応

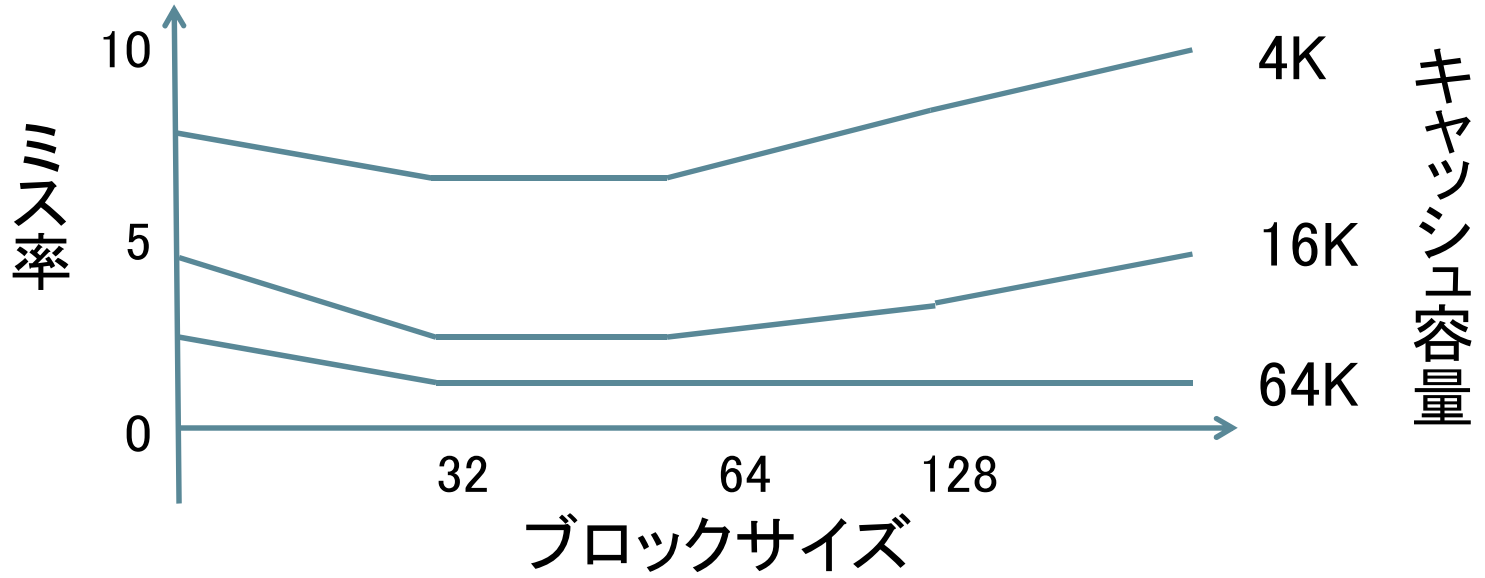
	有効	タグ	#0	#1	…	#15
0						
1						
63						

- ブロックサイズとヒット率

  大きなブロック⇒ミス率を下げられる（空間局所性の活用）

  反面，キャッシュ容量に対する相対的なブロック数を大きくするとミス率の上昇につながる

  また，ミス時のミスペナルティの増大にもつながる



- キャッシュミスの取り扱い
  1. 元のプログラムカウンタ値（現在のPC−4）をメモリに転送
  2. 主記憶から読み出しを行うよう指示，完了を待機
  3. キャッシュの該当するブロックに書き込みを行う．その際，主記憶から読み出したデータをキャッシュのデータ部分に格納し，アドレスの上位4ビットをALUからタグフィールドへ収め，有効ビットをON
  4. 実行命令を最初のステップから再開．

     （命令をフェッチしなおすことにより，キャッシュはヒットする）

- ライトスルーとライトバック
  - キャッシュ：読み出し性能の改善目的
    書き込み要求のためのものではない
  - ライト操作によって，元のメモリの値を変更が必要
  - メモリに転送を要求するだけでなく，キャッシュは当該データの有無を探索．もし存在する場合，その値も変更が必要
  - ライトスルーキャッシュ：
    キャッシュはコピーを保持．ライト操作をメモリに転送
  - ライトバックキャッシュ：
    キャッシュがローカルにデータを保持，必要時にメモリに値を書き込む．
    どのデータを書き戻すか⇒ダーティビット

- 書き込みの取り扱い

  ⇒キャッシュと主記憶の一貫性の保持

  ライト・スルー方式：

  キャッシュと主記憶に毎回書き込む方式

  例）メモリへの書き込み時間：CPUの１００サイクル分

  命令の10%がストア命令，元々CPUのCPIが1.0の場合

  CPI = 1.0＋100×10% = 11.0　性能が10分の1に低下

- 書き込み時の取り扱い
  - ライト・バッファ方式
    - 書き込み用のバッファを用意し，CPUはバッファへの書き込みで書き込み操作を完了
    - バッファから主記憶への書き込み速度が，CPUの書き込み派生頻度より低いと効果ない
  - ライト・バック方式
    - 書き込み発生時はキャッシュのみに書き込み
    - 置き換え対象になった時のみ，主記憶へ書き込み
    - 複雑な構造が必要

- ライトバックキャッシュの性能向上の例
  - メモリ内に値を増加させるプログラムにおけるループ
    - ライトスルーキャッシュ：
      - ループ実行ごとに，メモリ上のデータを更新する
    - ライトバックキャッシュ：
      - プログラム実行中は値をキャッシュ上に保持
      - ループ終了後，メモリ上のデータを更新

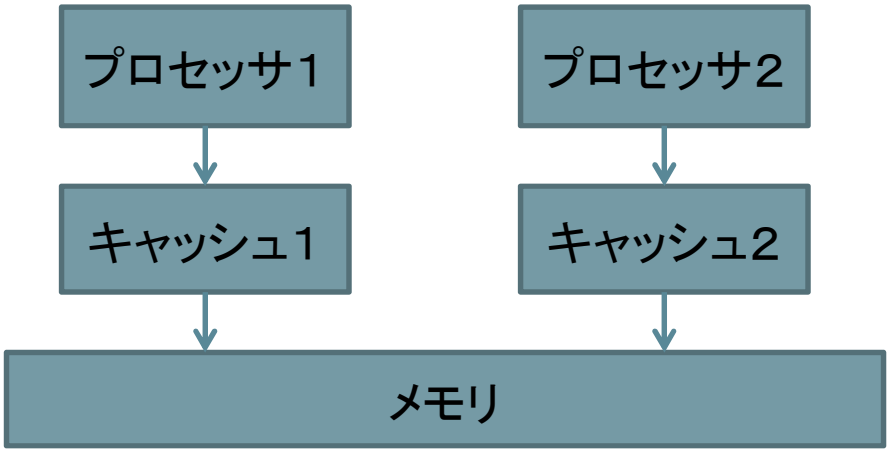
- キャッシュの一貫性（コヒーレンス）

  2つのプロセッサが，それぞれキャッシュを用いてメモリにアクセスする場合

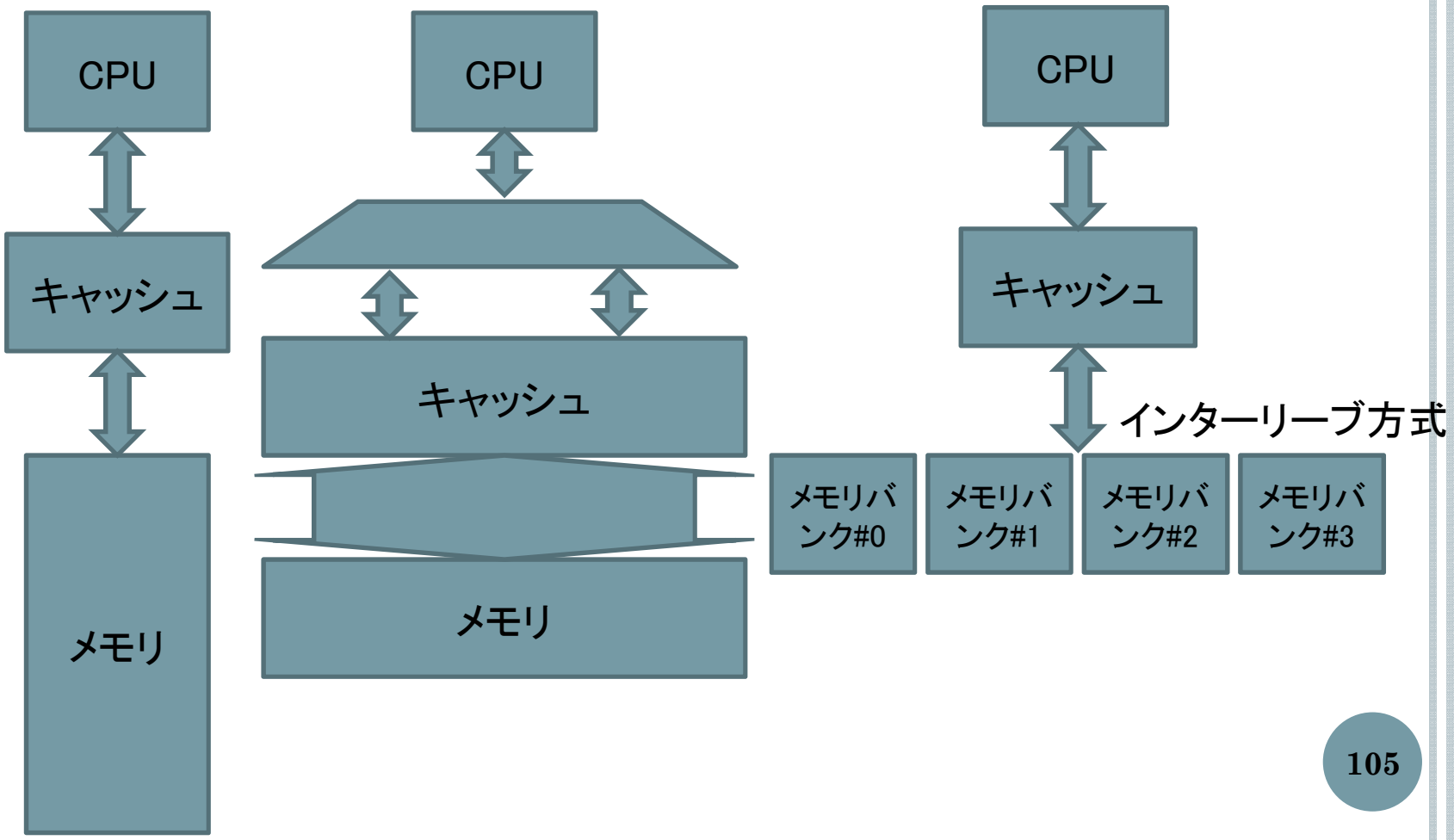
  ⇒ キャッシュの一貫性プロトコル(ハードウエアの追加)

  プロセッサ2がアドレスAからデータを読むとき，一貫性プロトコルは，キャッシュ2にキャッシュ1に通知を要求

  キャッシュ1がアドレスAのデータを保持している場合，キャッシュ1はデータを最新のものに更新



- キャッシュを支援する記憶システム



キャッシュを支援する記憶システム

キャッシュミス発生時：必要な語は主記憶から読み出し

例）アドレス送出：1メモリバスクロックサイクル

DRAMの一語当たりのアクセス時間：15メモリバスクロックサイクル

データの一語の転送：1メモリバスクロックサイクル

キャッシュのブロックは4語から構成

DRAMのバンク幅が1語の場合ミスペナルティ

1+4×15+4×1=65 メモリバスクロックサイクル

メモリのデータ幅を2語長

1+2×15+2×1=33 メモリバスクロックサイクル

バンク数4のメモリ構成（インターリーブ）

1+1×15+4×1=20 メモリバスクロックサイクル

メモリストールとCPU時間

CPU時間＝（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

キャッシュミスの増大⇒メモリストールクロック数の増大

メモリストールクロック数＝読み出しストールクロック数＋書き込みクロックストール数

読み出しストールクロック数＝プログラム当たりの読み出し件数×読み出しミス率×読み出しミスペナルティ

書き込みストールクロック数＝プログラム当たりの読み出し件数×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

メモリストールとCPU時間

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件数×ミス率×ミスペナルティ

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命令件数×1メモリアクセス命令当たりのミス率×ミスペナルティ

例題1

あるコンピュータ
　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%
　プロセッサのCPI：メモリのストールなしで2
　ミスペナルティ＝すべてのミスに対して100クロックサイクル

ミスのないプロセッサに対して，このコンピュータはどの程度の速度となるか．ただし，メモリアクセス命令の出現頻度は36%に想定

解答例

命令数をＩとすると，

　命令のミスクロック数＝I×2%×100＝2.0 I

メモリアクセス命令数は36%なので

　データのミスクロック数＝I×36%×4%×100＝1.44 I

よって1命令当たりのメモリストールの合計クロック数は3.44

以上より

$$\frac{\text{メモリストールのあるCPU時間}}{\text{完全キャッシュを備えたマシンのCPU時間}} = \frac{2 + 3.44}{2} = \frac{5.44}{2}$$

よってメモリストールがあると，完全なキャッシュを備えるコンピュータに比べその性能は2.72分の1となる

例題2

例題1とクロック周波数も含め同一条件下でプロセッサを高速なものにした場合どうなるか

プロセッサの速度を例1の2CPIのものから，その速度を2倍に向上させCPIが1になったとする．この場合，メモリストールに対する合計のクロック数は3.44と変化はないので

$$\frac{\text{メモリストールのあるCPU時間}}{\text{完全キャッシュを備えたマシンのCPU時間}} = \frac{1 + 3.44}{1} = \frac{4.44}{1}$$

となる．この場合，メモリストールに要する時間の割合は，3.44/5.44＝63%から3.44/4.44＝77%へ増大することになる

キャッシュミスの影響

例題2で示したように，記憶システムを変えずにプロセッサの速度のみを向上させると，キャッシュミスによる性能低下を大きくする．

このことは，記憶システムを変えずにクロック周波数を引き上げても同様に，キャッシュミスによる性能低下を大きくする．

また，ヒット時間が大きくなると，記憶システムからの語アクセスに要する合計時間が長くなり，結果としてプロセッサのクロックサイクル時間が増大する可能性がある．このことは，キャッシュ容量を大きくした場合に，注意が必要である．

⇒キャッシュ容量を単に増大するのではなく，多段階のキャッシュの構成につながる

- L1,L2,L3キャッシュ

  多くのコンピュータメモリシステム

  ⇒　2レベル以上のキャッシュ階層

  背景

  1. 伝統的なメモリキャッシュは，メモリ，プロセッサ双方から独立していた
  2. キャッシュへのアクセスには，プロセッサチップと接続する接続する信号線が必要
  3. 外部ハードウエアに信号線を使うのは，チップ内の機能ユニットにアクセスするのに比べ，アクセス遅延大
  4. 半導体技術の進歩により，チップ内に搭載できるトランジスタ数増大

  ⇒プロセッサチップ内に2次キャッシュ搭載

  L1キャッシュ：プロセッサチップ内（オンチップ）

  L2,L3キャッシュ：プロセッサチップ外（オフチップ）

平均メモリアクセス時間 AMAT

ヒットした場合とミスした場合の両方を考慮したメモリアクセス時間の平均値

AMAT＝ヒットした場合のアクセス時間＋ミス率×ミスペナルティ

クロックサイクル時間が1ns，ミスペナルティが20クロックサイクル．命令当たりのミス率が0.05，キャッシュへのアクセス時間が1クロックサイクルであるプロセッサのAMATはいくらか．ただし，読み出しと書き込みのミスペナルティは等しいものとし，その他の書き込みストールは無視する．

AMAT＝1＋0.05×20＝2 クロックサイクル，すなわち，2nsとなる

柔軟性の高いブロックの配置によるミスの削減

- ダイレクトマッピング方式

  メモリ・ブロックを配置するキャッシュの場所が特定

- フル・アソシアティブ方式

  メモリ・ブロックを配置するキャッシュの場所が任意

- セット・アソシアティブ方式

  メモリ・ブロックを配置するキャッシュの場所が，あるきまった数 n（セット数）に定められている

  ⇒ nウエイ セット・アソシアティブ方式

ダイレクトマッピング方式⇔ 1ウエイ セットアソシアティブ方式

フル・セットアソシアティブ方式（キャッシュがm個のブロック）

⇔ 1ウエイ セットアソシアティブ方式

連想度：1セットのブロック数

- ダイレクトマッピング方式におけるブロックの場所
  ブロック番号をキャッシュ内のブロック数で割った剰余
- フル・アソシアティブ方式
  キャッシュ内の任意の位置にブロックを配置
  ブロックの位置：キャッシュ内のすべての要素の探索が必要
- セット・アソシアティブ方式におけるブロックが含まれるセットの位置
  ブロック番号をキャッシュ内のセット数で割った剰余
  ブロックの位置：セット内のすべての要素の探索が必要

アドレス12のブロックが格納される（可能性のある）キャッシュ内の位置
キャッシュは8ブロック

8ブロックのキャッシュがとりうる形態

ブロック	タグ	データ
0		
1		
2		
3		
4		
5		
6		
7		

ダイレクトマッピング方式

セット	タグ	データ	タグ	データ
0				
1				
2				
3				

2ウエイ セットアソシアティブ方式

ほかに8ウエイセットアソシアティブ
（フルアソシアティブ方式）がある

セット	タグ	データ	タグ	データ	タグ	データ	タグ	データ
0								
1								

4ウエイ セットアソシアティブ方式

キャッシュにおける連想度とミス

- セットアソシアティブ方式

  連想度を増やす利点⇒ミス率の低減

  その欠点⇒ヒット時間の増大

例題）連想度とミス

1語のブロック4つからなるキャッシュを想定し，ブロックアドレスが0，8，0，6，8の順にアクセスするとき，以下の方式におけるキャッスミスの発生数

① フルアソシアティブ方式
② 2ウエイセットアソシアティブ方式
③ ダイレクトマッピング方式

ダイレクトマッピング方式

各ブロックアドレスとキャッシュブロックの対応

ブロックアドレス	キャッシュブロック
0	0 mod 4 = 0
6	6 mod 4 = 4
8	8 mod 4 = 0

各ブロックアドレスを参照した後のキャッシュの内容

参照したメモリブロックのアドレス	ヒット/ミス	参照後の各キャッシュブロックの内容			
		0	1	2	3
0	ミス	メモリ[0]			
8	ミス	メモリ[8]			
0	ミス	メモリ[0]			
6	ミス	メモリ[0]		メモリ[6]	
8	ミス	メモリ[8]		メモリ[6]	

セット・アソシアティブ方式

各ブロックアドレスとキャッシュブロックの対応

ブロックアドレス	キャッシュのセット
0	0 mod 2 = 0
6	6 mod 2 = 0
8	8 mod 2 = 0

セット内はLRU(least recently used)
により置換ブロックを決定

各ブロックアドレスを参照した後のキャッシュの内容

参照したメモリブロックのアドレス	ヒット/ミス	参照後の各キャッシュブロックの内容			
		セット0	セット0	セット1	セット1
0	ミス	メモリ[0]			
8	ミス	メモリ[0]	メモリ[8]		
0	ヒット	メモリ[0]	メモリ[8]		
6	ミス	メモリ[0]	メモリ[6]		
8	ミス	メモリ[8]	メモリ[6]		

フル・アソシアティブ方式

各ブロックアドレスとキャッシュブロックの対応

各ブロックアドレスを参照した後のキャッシュの内容

参照したメモリブロックのアドレス	ヒット/ミス	参照後の各キャッシュブロックの内容			
		ブロック0	ブロック1	ブロック2	ブロック3
0	ミス	メモリ[0]			
8	ミス	メモリ[0]	メモリ[8]		
0	ヒット	メモリ[0]	メモリ[8]		
6	ミス	メモリ[0]	メモリ[8]	メモリ[6]	
8	ヒット	メモリ[0]	メモリ[6]	メモリ[6]	

連想度とミス率

連想度とミス率の関係を示す実験結果

1ブロック16語からなる64Kバイトのデータキャッシュを例

連想度	ミス率
1	10.3%
2	8.6%
4	8.3%
8	8.1%

キャッシュ内のブロックの見つけ方

- セット・アソシアティブ方式

  キャッシュ中の各ブロックには，そのブロックのアドレスを示すアドレスタグを付加

タグ	インデックス	ブロック内のオフセット

アドレスの3つの部分

インデックスはセットの選択に，タグはセット中の全ブロックと比較してブロックを選択するために使用される．
ブロック内オフセットはブロック中の求めるデータのアドレス

セット中の全ブロックの探索は並列的に実行される

キャッシュの全容量を一定に保つ場合

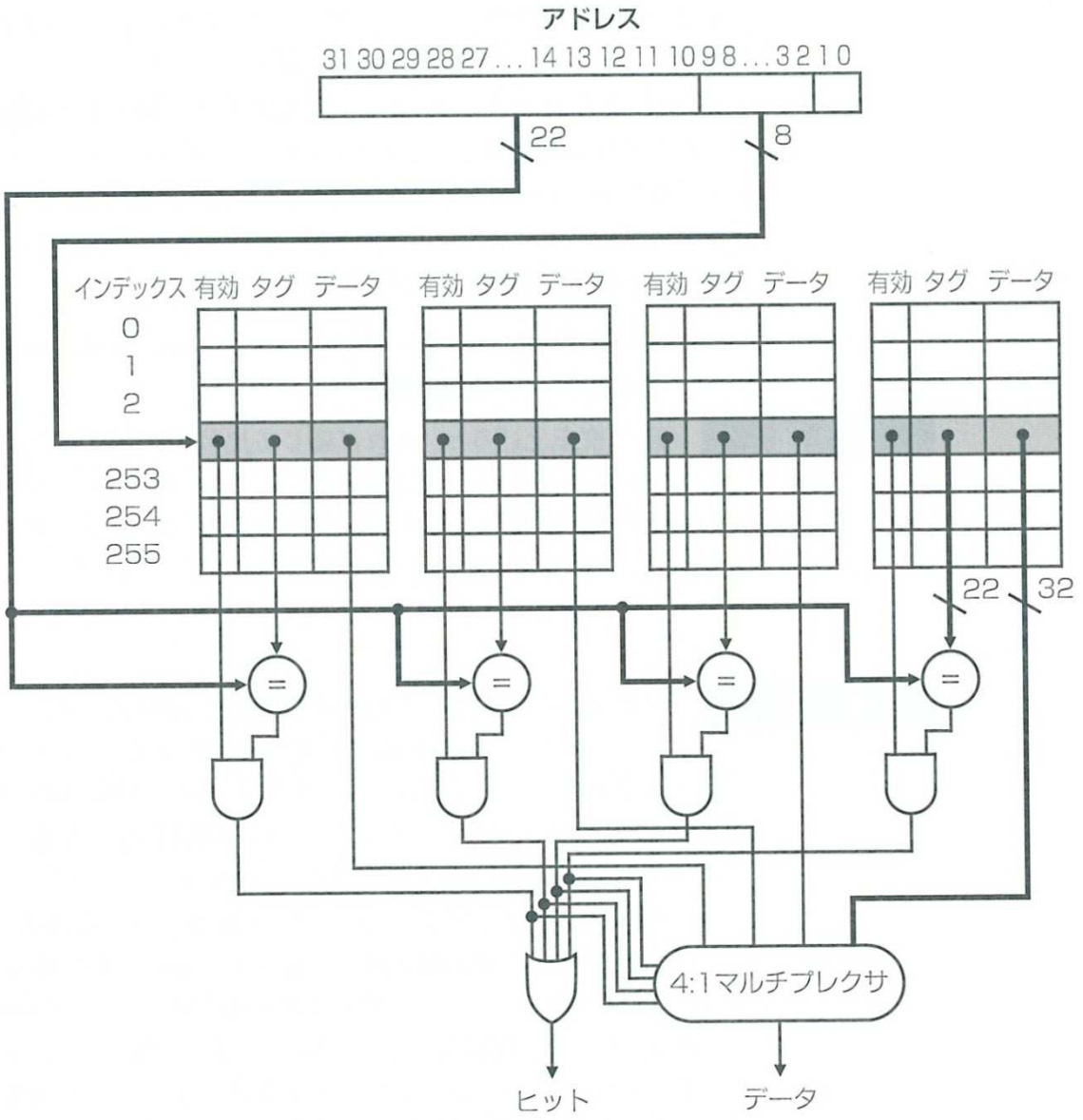
連想度（1セット当たりのブロック数）を2倍に増やすと，セット数は半分に減少

⇒インデックスは1ビット減少し，タグ長が1ビット増加

⇒フルアソシアティブ方式：セット数は1（インデックスは不要）

すべてのブロックを並列的に照合が必要性

4ウエイセットアソシアティブ方式



置き換え対象ブロックの選択

- ダイレクトマッピング方式

  ブロックの格納場所は1つ

- アソシアティブ方式

  ブロックの格納場所を選択可能

  ⇒どのブロックを置き換えるかを決定する必要がある

- 一般的な方法LRU法

  使用されずにいた時間が最も長いブロックを選択

  2ウエイセットアソシアティブ方式の場合，要素が参照されるたびにどちらが使用されたか記録　⇒ 1ビット

タグのサイズと連想度

連想度を上げるとそれに応じて比較器が増加するとともに，キャッシュブロック当たりのタグのビット数が増加．4Kブロックのキャッシュがあり，そのブロックサイズが4語である．またそのアドレスは32ビットとする．ダイレクトマッピング方式，2ウエイおよび4ウエイセットアソシアティブ方式，フルアソシアティブ方式のキャッシュについて，セットの総数とタグビットの総数を求めよ．

ブロック当たりのバイト数は2^4＝16

アドレス長が32ビット

　⇒インデックスとタグに32−4＝28ビット使用

ダイレクトマッピング方式

　セット数＝ブロック数

　4K= 2^{12}より，インデックスは12ビット

　タグの総数は（28−12）×4K=64K

- 2ウエイセットアソシアティブ方式
  - 連想度を1つ上げると，セット数が半分になる
  - インデックスが1ビット減り，タグ中のビット数が1ビット増加
  - セット数は2K
  - タグビットの総数 (28−11)×2×2K=68K ビット
- 2ウエイセットアソシアティブ方式
  - セット数は1K
  - タグビットの総数 (28−10)×4×1K=72K ビット
- フルアソシアティブ方式
  - セット数は1つ，ブロック数は4K
  - タグの総ビット数は28×4K=112Kビット

- キャッシュとしてのTLB（変換側付きバッファ）
- デマンドページングシステムで利用されているTLB

  劇的にデマンドページングシステムの性能を向上させている
- 小規模かつ高速なハードウエア機構から構成
- TLB：キャッシュそのもの

マルチレベルキャッシュ

- DRAMにアクセスに要する時間と，CPUのクロック周波数とのギャップの解消のため
- CPUと同一のチップ上に，2次キャッシュを実装

- L1,L2,L3キャッシュの容量

プロセッサ	L1キャッシュ	L2キャッシュ	L3キャッシュ
Itanium2	32KB	256KB	3MB,4MB or 6MB
Itanium	32KB	96KB	2MB or 4MB
Xeon MP	8KB	256KB or 512KB	512KB,1MB or 2MB
P4	8KB	512KB	—

マルチレベルキャッシュの性能

基本CPIが1.0のCPU，クロック周波数は4GHz.
主記憶へのアクセス時間は，キャッシュミスに関する処理も含め100nS．1次キャッシュにおける命令あたりのミス率は2%.
2次キャッシュを追加したとき，それへのアクセス時間は，5ns．2次キャッシュは，主記憶へのミス率を0.5%に下げられるだけの容量があると仮定.
CPUの速度の向上はどの程度か

主記憶へのミスペナルティは

100ns÷0.25ns/クロックサイクル＝400クロックサイクル

キャッシュが1レベルの場合，実行CPIは

実行CPI＝基本CPI＋命令あたりのメモリストールサイクル数

＝1.0＋2%×400＝9.0

2次キャッシュを追加すると，2次キャッシュに対するミスペナルティは

5ns÷0.25ns/クロックサイクル＝20クロックサイクル

2次キャッシュにより主記憶へのミス率は0.5%となるので，

実行CPI＝1.0＋2%×20＋0.5%×400＝3.4

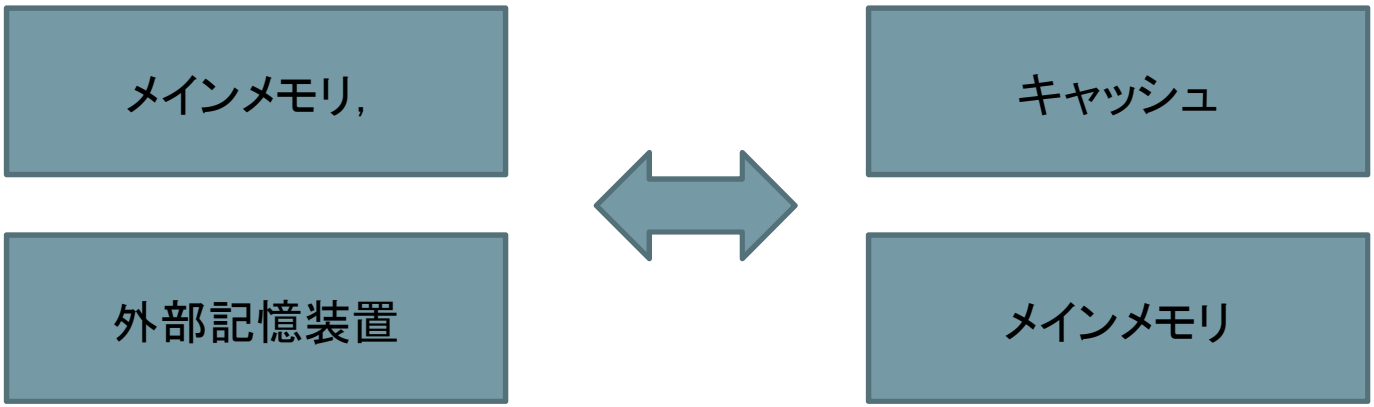
2次キャッシュを参照するだけで済んだ，ストールサイクル数＋主記憶までアクセスしたときのストールサイクル数（2次キャッシュへのアクセスも加算）

（2%－0.5%）×20＝0.3，0.5%×（20＋400）＝2.1

1.0+0.3+2.1＝3.4

- マルチレベルキャッシュ
  - 単一レベルキャッシュに比べ，
    - 1次キャッシュ：
      - ミスペナルティの低減がねらい
      - 容量は小さく，ブロックサイズも小さい
    - 2次キャッシュ：
      - ミス率の低下が目的
      - 容量は大きく，より大きなブロックサイズ
      - 1次キャッシュに比べ，連想度も高い

- キャッシュ技術としてのデマンドページング
- 概念的にキャッシュ技術の一つの形



デマンドページング

仮想空間をメインメモリより広くとることができる

キャッシュシステム

キャッシュはページ全体の一部を保持

- 仮想アドレス使用
  - MMUが仮想アドレスを物理アドレスに変換前にキャシュが応答可能⇒メモリ応答速度向上
  - MMUがプロセッサチップ外にある場合，L1キャッシュは仮想アドレスを使わねばならない
  - キャッシュが仮想メモリシステムと相互に作用することを可能とするハードウエアの追加が必要

- 仮想メモリキャッシュ技術とキャッシュフラッシュ

  キャッシュ技術と仮想メモリの併用時：

  キャッシュは，プロセッサとMMUの間？

  MMUと物理メモリの間？

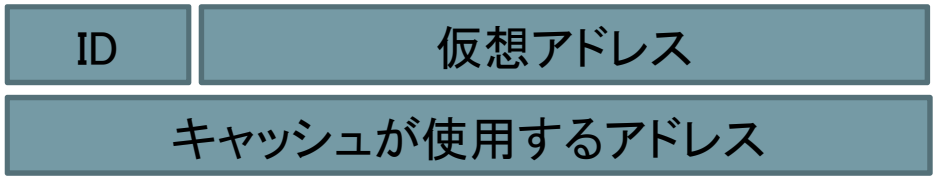
  キャッシュのデータを指定するとき，仮想アドレスか，

  物理アドレスか

- 仮想メモリシステムが，通常アプリケーションプログラムに同一アドレス空間を提供時
  - アプリケーションプログラムは0番地から開始
  - OSがアプリケーションをスイッチする時
  - アプリケーションは新しい値を参照するのに同じアドレスを使用 → キャッシュのデータ取り替え必要
- 複数のアプリケーションが同一アドレスを使用時の，あいまい性の克服方法
  - キャッシュフラッシュ命令

    OSが新しい仮想アドレス空間に変わるごとにキャッシュをフラッシュ
  - あいまい性を排除した認証

    アドレス空間を認証するためのビットを使用



- プログラマにとっての重要性
  - プログラム中のループ：繰り返し小さな命令集合へのアクセス
  - 同じデータの参照

大規模配列の各要素に，何度も繰り返し処理するプログラム

- 次の要素に移行する前に，配列の一要素にすべての演算を実行する
- ⇒ その要素がキャッシュに残っているので，高速処理が可能

- 命令とデータキャッシュ

  命令：連続性が高く，高い局所性

  データ：ランダム性があり，局所性は低い

  ランダムな参照を連続したアクセスに挿入すると，キャッシュの性能を悪化

  ランダムな参照数を低減させることで，キャッシュ性能は向上