7 キャッシュと仮想記憶

♣ キャッシュの詳細

♣ 仮想記憶の詳細

♣ プログラミングやアルゴリズムとの関係

7.1 キャッシュ

【復習】キャッシュ (cache)

- CPU と主記憶の間にある, 高 速で 小 容量のメモリー
- 主記憶にアクセスのあったデータを保存し, 同じアクセスがあった場合に再利用
  → 主記憶へのアクセスを (見かけ上) 高速 化する
- アクセスしたデータがキャッシュにある → ヒット ; ない → ミス

7.1.1 キャッシュブロックとキャッシュの動作

- キャッシュ ブロック (cache block) (キャッシュライン (cache line) とも言う)
  - キャッシュでは, 1 バイト毎ではなく, 何バイトかをまとめた「ブロック」単位でデータを管理する
  - 一般的なブロックのサイズは 32B ~ 256B 程度

  (例) 1 ブロック 64B の場合

		オフセット (offset; ブロック内での位置)
ブロック 0 番	0 番地	0
	1 番地	1
	2 番地	2
	...	...
	63 番地	63
ブロック 1 番	64 番地	0
	65 番地	1
	66 番地	2
	...	...
	127 番地	63
ブロック 2 番	128 番地	0
	...	...

  このとき, 76 番地のブロック番号 = 1 (76/64), オフセット = 12 (76%64)

- キャッシュの動作 (読み出し)

  主記憶の a 番地の読み出し (ブロック b, オフセット f とする)

  (1) ブロック b がキャッシュ内にあれば (ヒット) (2) へ; なければ (ミス):

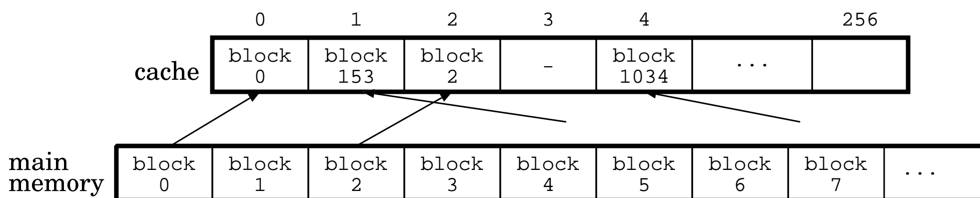
  * ブロック b を 主記憶 からキャッシュにロード (転送)

    ただし, この際にブロック b をロードする場所が満杯なら, 先に他のブロックを 追い出 して空きを作る

  (2) ブロック b のオフセット f のデータを CPU に返す

7.1.2　キャッシュの写像方式

☆ 主記憶の b 番ブロックをキャッシュの どこ に格納するか, ということ



(1) フルアソシアティブ方式 (full associative ; 完全連想方式) [1]

- b 番ブロックのキャッシュ内での位置は 任意
- キャッシュの利用効率は非常に 高い
- 自由度が高すぎてブロックの検索が 困難 で , ハードウェアが 複雑 になりすぎる

☆ 実際のキャッシュでは用いられない

(2) ダイレクトマッピング方式 (direct mapping ; 直接写像方式)

- b 番ブロックのキャッシュ内での位置を 一意 にする
- ブロックの検索が 容易 で, ハードウェアも 単純
- キャッシュの利用効率が 悪い (キャッシュ内で 衝突 が起き易い)

(3) セットアソシアティブ方式 (set associative; セット連想方式)

- b 番ブロックのキャッシュ内での位置は S 個の どれか
  - S-way , あるいは連想度 S と言う[2]
- S を調節して利用効率と検索の容易性の バランス を取る
  - $S =$ 1 とすればダイレクトマッピングになる
  - $S =$ キャッシュ容量 とすればフルアソシアティブになる
  - S は通常 2 か 4 か 8 程度

7.1.3　ブロック置き換えアルゴリズム

☆ キャッシュ内での衝突時に, どのブロックを追い出すか[3]

(1) ランダム (random)

(2) FIFO (first-in first-out) ··· 一番 最初 に入ったブロックを追い出す

(3) LRU (least recently used) ··· 最後のアクセスが最も 昔 のブロックを追い出す

[1] ここでは, associative を「連想」と訳すと分かり難くなる．意訳になるが「自由」の方が理解し易い.

[2]「連想度」の元英語は associativity. これも「自由度」と意訳すればわかり易い.

[3] 直接写像方式ではブロックの入る場所が一意なので選択の余地はないが, フルアソシアティブ方式やセットアソシアティブ方式では, どのブロックを追い出すか選択しなければならない.

7.1.4 キャッシュの動作の詳細とヒット率の計算

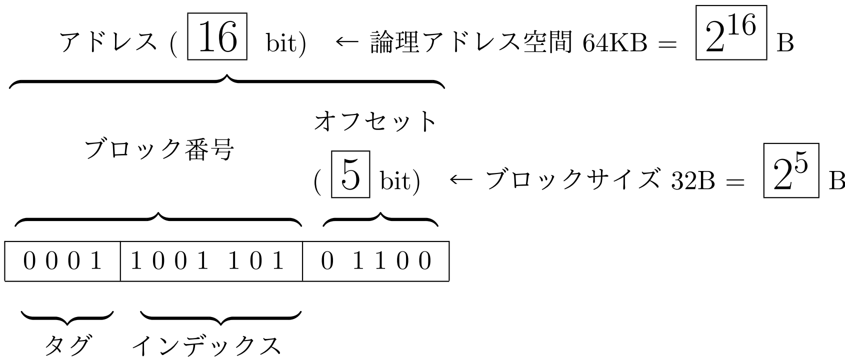
(1) フルアソシアティブ方式　現実のキャッシュで用いられることはないので省略

(2) ダイレクトマッピング方式

例題 7.1 論理アドレス空間 64KB のコンピュータにおいて, 下記の 1)~8) の順に主記憶の番地にアクセスがあったとする. このコンピュータに容量 4KB, ブロックサイズ 32B のダイレクトマッピング方式のキャッシュが搭載されているとき, キャッシュミスは何回発生するか. また, この場合のキャッシュヒット率は何%か.

```
1) x8034  (b 1000 0000 0011 0100)
2) x8038  (b 1000 0000 0011 1000)
3) x09AC  (b 0000 1001 1010 1100)
4) x9024  (b 1001 0000 0010 0100)
5) x09A8  (b 0000 1001 1010 1000)
6) x803C  (b 1000 0000 0011 1100)
7) x9028  (b 1001 0000 0010 1000)
8) xD03C  (b 1101 0000 0011 1100)
```

(解) アドレスのビット列は, タグ , インデックス , オフセット に分けられる



(7 bit) ← 容量 $4\text{KB}/32\text{B} = 2^{12}/2^{5} = 2^{7}$ ブロック

– オフセット ··· キャッシュブロック内での位置 (無視できる)

– インデックス (index) ··· このブロックをキャッシュのどの位置に入れるか (ここが重要)

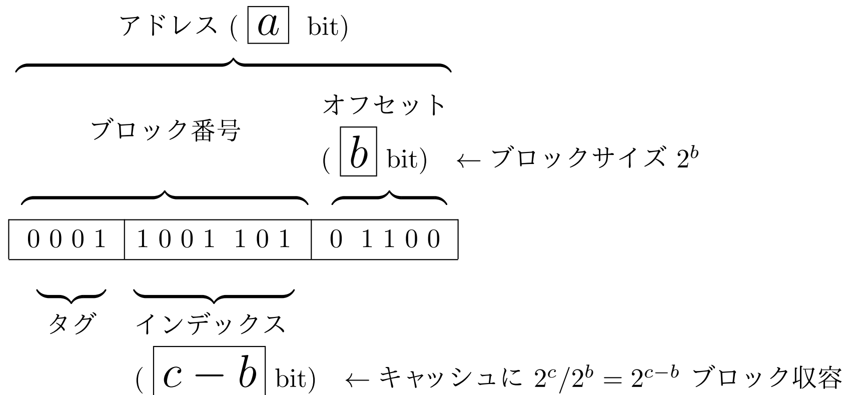
– タグ (tag) ··· インデックが同じブロックの識別に用いる

キャッシュの内容は次のように変化する

アドレス tag index offset	hit/miss	アクセス後のキャッシュの内容 (収容されるブロックのタグ)					
		0000 000 (0)	0000 001 (1)	···	1001 101 (77)	···	1111 111 (127)
1) 1000 0000 0011 0100	miss		1000				
2) 1000 0000 0011 1000	hit		〃				
3) 0000 1001 1010 1100	miss		〃		0000		
4) 1001 0000 0010 0100	miss		1001		〃		
5) 0000 1001 1010 1000	hit		〃		〃		
6) 1000 0000 0011 1100	miss		1000		〃		
7) 1001 0000 0010 1000	miss		1001		〃		
8) 1101 0000 0011 1100	miss		1101		〃		

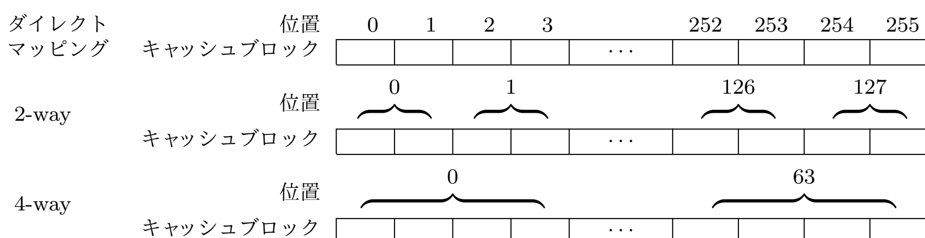
キャッシュミスは 6 回発生し, キャッシュヒット率は 2/8 = 25 %

- 一般の場合 (ダイレクトマッピング方式)
  - 論理アドレス空間が $A = \boxed{2^a}$ バイト (アドレスが $\boxed{a}$ bit で指定される)
  - キャッシュブロックサイズが $B = \boxed{2^b}$ バイト
  - キャッシュ容量が $C = \boxed{2^c}$ バイト



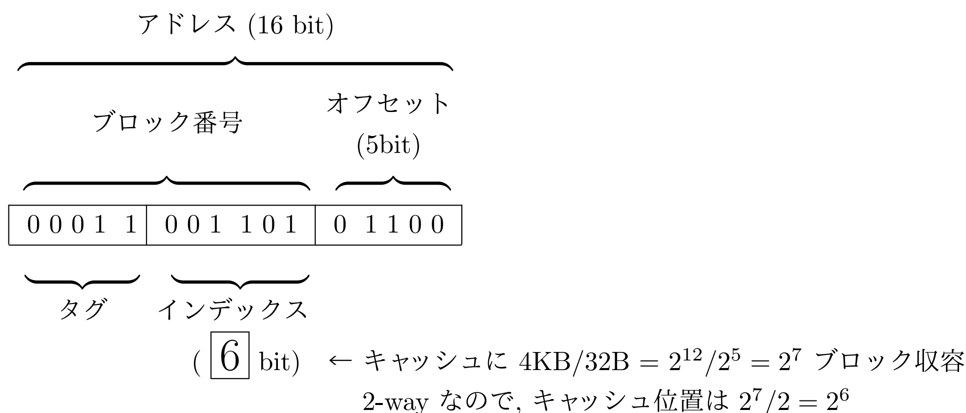
(3) セットアソシアティブ方式

- S-way の場合
  - キャッシュ各位置に $\boxed{S}$ 個のブロックを収容できる
  - キャッシュの位置の数は, 収容可能ブロック数 / $\boxed{S}$ になる



例題 7.2 論理アドレス空間 64KB (= 2^{16}B) のコンピュータにおいて, 例題 7.1 と同じメモリーアクセスがあったとする. このコンピュータに容量 4KB, ブロックサイズ 32B の2-way のセットアソシアティブ方式のキャッシュが搭載されているとき, キャッシュミスは何回発生するか. また, この場合のキャッシュヒット率は何%か. ただし, ブロック置き換えアルゴリズムは FIFO であるとする.

(解) アドレスは 16bit であり, タグ, インデックス, オフセットは下記のようになる.

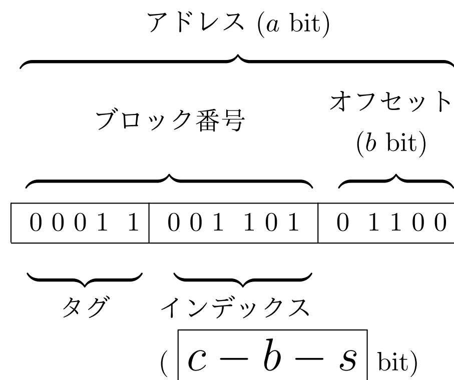


キャッシュの内容は次のように変化する

アドレス tag index offset	hit/miss	アクセス後のキャッシュの内容 (収容されるブロックのタグ) 000 000 (0)	000 001 (1)	···	001 101 (13)	···	111 111 (63)
1) 1000 0000 0011 0100	miss		1000 0				
2) 1000 0000 0011 1000	hit		〃				
3) 0000 1001 1010 1100	miss		〃		0000 1		
4) 1001 0000 0010 0100	miss		1000 0 1001 0		〃		
5) 0000 1001 1010 1000	hit		〃		〃		
6) 1000 0000 0011 1100	hit		〃		〃		
7) 1001 0000 0010 1000	hit		〃		〃		
8) 1101 0000 0011 1100	miss		1001 0 1101 0		〃		

キャッシュミスは 4 回発生し, キャッシュヒット率は 4/8 = 50 %

- 一般の場合 (セットアソシアティブ方式)
  - 論理アドレス空間が $A = 2^a$ バイト
  - キャッシュブロックサイズが $B = 2^b$ バイト
  - キャッシュ容量が $C = 2^c$ バイト
  - S-way ($S =$ 2^s)



7.2 仮想記憶

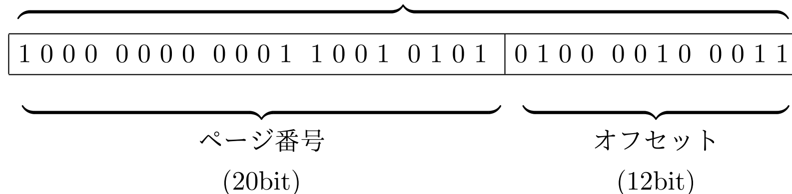
【復習】仮想記憶 (virtual memory) とは

主記憶と補助記憶装置の組合せにより, 仮想的に巨大な主記憶が利用できるようにする技術

- 用語
  - 仮想アドレス (virtual address) ··· 仮想 (的な巨大) 記憶での番地 (論理 アドレス)
  - 実アドレス (real address) ··· 現実の主記憶での番地 (物理 アドレス)
  - ページ (page) ··· データの入れ換えの単位 (キャッシュの ブロック に相当)
    - ∗ 一般的なページのサイズは 4KB 程度 (最近では 512KB ∼ 4MB のものもある)
  - ページテーブル (page table) ··· 仮想アドレスと実アドレスの 対応 を記憶する表
  - ページフォルト (page fault) ··· アクセスしたページが 主記憶 にないこと
    - ∗ 割り込み が発生して, OS がページの 入れ換え を行う
  - スラッシング (thrashing)
    - ∗ 仮想記憶への過負荷によって, CPU/OS が ページング に処理能力のほとんどを費やす状態

- ページ番号の計算方式 → キャッシュのブロック番号と同じ

(例) 論理アドレス空間 2^{32}B, ページサイズ 4KB (=2^{12}B) の場合



- 写像方式には, 通常 フルアソシアティブ 方式が用いられる

例題 7.3 主記憶に仮想記憶の 4 ページを収容できるコンピュータ[4] において,

0 1 7 8 4 1 7 6 1 7 4 2 4 5 2 6 3 4

という (仮想) ページ参照系列があったとき, FIFO と LRU でそれぞれページフォルトは何回発生するか?

FIFO の場合

参照ページ	ページフォルト	主記憶の内容 (新しく入った物が右)
0	√	0
1	√	0, 1
7	√	0, 1, 7
8	√	0, 1, 7, 8
4	√	1, 7, 8, 4
1		1, 7, 8, 4
7		1, 7, 8, 4
6	√	7, 8, 4, 6
1	√	8, 4, 6, 1
7	√	4, 6, 1, 7
4		4, 6, 1, 7
2	√	6, 1, 7, 2
4	√	1, 7, 2, 4
5	√	7, 2, 4, 5
2		7, 2, 4, 5
6	√	2, 4, 5, 6
3	√	4, 5, 6, 3
4		4, 5, 6, 3

ページフォルトは 13 回

LRU の場合

参照ページ	ページフォルト	主記憶の内容 (新しくアクセスしたものが右)
0	√	0
1	√	0, 1
7	√	0, 1, 7
8	√	0, 1, 7, 8
4	√	1, 7, 8, 4
1		7, 8, 4, 1
7		8, 4, 1, 7
6	√	4, 1, 7, 6
1		4, 7, 6, 1
7		4, 6, 1, 7
4		6, 1, 7, 4
2	√	1, 7, 4, 2
4		1, 7, 2, 4
5	√	7, 2, 4, 5
2		7, 4, 5, 2
6	√	4, 5, 2, 6
3	√	5, 2, 6, 3
4	√	2, 6, 3, 4

ページフォルトは 11 回

[4] 簡単のため 4 ページにしているが, 実際のコンピュータではそんなに少なくない. 念のため.

7.3 キャッシュミス/ページフォルト時の処理の比較

	キャッシュ	仮想記憶
処理を行うのは	ハード ウェア	ソフト ウェア (OS)
割り込みは	発生 しない	発生 する
処理の間 CPU は	完了を 待つ	他 のプログラムを実行

☆ 各装置の速度から理解できる

CPU (数百 ps∼ 数 ns) $\simeq$ キャッシュ (1∼10ns) $<$ 主記憶 (50∼100ns) $\ll$ HDD (数 ms)

7.4 プログラミングやアルゴリズムとの関係

- アルゴリズム的に高速な計算法が, 実際には 遅い こともある
- 遅くなるアルゴリズム/プログラムの例
  - 広い範囲を ランダム にアクセスする計算は概して遅い
  - 大きなデータ構造を繰り返し 走査 する計算は概して遅い

    → 狭い 範囲で計算を繰り返すように書き直すと, 速くなることがある
- キャッシュを意識してプログラムを高速化する技法
  - 後でアクセスするデータがキャッシュから 追い出 されるのを防ぐ

    → ダミー のメモリーアクセスを挿入
  - 配列/構造体, 関数の番地を 調整 する
    - データ/コードがキャッシュブロックの境界を またぐ のを防ぐ

      (そのデータ/コードが使用するキャッシュブロック 数 が減る)
    - キャッシュ内収容位置の 衝突 を防ぐ

    → リンカーに番地を指示, 宣言の 順序 変更, ダミー 変数の宣言, etc.

Nagisa ISHIURA