■6 群（コンピュータ･基礎理論とハードウェア）- 4 編（コンピュータアーキテクチャ（I）標準的）

4 章 記　憶

【本章の構成】

本章では，記憶階層（4-2 節），キャッシュ（4-3 節），仮想記憶（4-4 節）について，それぞれ述べる．

なお，記憶デバイス（4-1 節）については，9 群 3 編「半導体」，及び，8 群 2 編「情報ストレージ」を参照して頂きたい．

■6 群 - 4 編 - 4 章

4-1 記憶デバイス

9 群 3 編「半導体」 参照.
8 群 2 編「情報ストレージ」 参照.

■6 群 - 4 編 - 4 章

4-2 記憶階層

（執筆者：津邑公暁）[2011 年 8 月 受領]

4-2-1 記憶装置の階層構造

記憶装置はコンピュータシステムの重要な構成要素である．特にフォン・ノイマン型コンピュータでは，データのみならずプログラムも記憶装置に格納されるため，記憶装置の構成およびアクセス速度はコンピュータシステム全体の性能に大きな影響を及ぼす．

高い性能のために記憶装置に求められる特徴は，大容量性と高速性である．しかし，記憶素子は一般に大容量になるほどアクセス速度が低下するため，これを両立することは不可能である．そこで，高速かつ小容量である記憶（キャッシュ，Cache）と，低速かつ大容量である記憶（仮想記憶）を階層構造に組み合わせることでこの 2 つの要求に応える．これを記憶階層と呼ぶ．

4-2-2 記憶参照の特徴

ではなぜ，このような階層構造によって大容量性と高速性が両立可能となるのであろうか．それは，プログラムによる記憶参照に，以下の 2 つの性質があるためである．

一つは，参照されたロケーション（アドレス，Address）が，近い将来に再度参照される可能性が高いという性質であり，これを時間的局所性（Temporal Locality）と呼ぶ．命令参照に関して言えば，プログラムは一般にサブルーチンやループを多く用いて構成されるが，それらは繰り返し実行されるため，同一アドレスが何度も参照される．データについても，ある変数に値を代入した後，その変数から値を読み出して演算するなどの操作は頻繁に行われる．

もう一つは，参照されたアドレスの近辺が次に参照される可能性が高いという性質であり，これを空間的局所性（Spatial Locality）と呼ぶ．例えば，ある配列を構成する要素はメモリ空間上に連続して配置されるため，これらを順に処理する場合などにこの性質が現れる．また，プログラムを構成する命令列は分岐等が発生しない場合は順次参照されるため，命令参照にも同様の性質が見られる．

これら 2 つの特徴から，一度参照されたアドレスをより高速な記憶装置にコピーしておくことで，再度同じアドレスへのアクセスを高速化できる．また，高速な記憶装置にコピーする際に，ある程度の大きさを持ったブロック単位でコピーしておくことで，今後発生するであろう，そのアドレスの近辺へのアクセスも高速化できる．

キャッシュは，主記憶上のデータのうち，このような今後アクセスされる可能性の高いデータを格納しておくための高速・小容量な記憶装置である．一方で，仮想記憶は，大規模な記憶領域を実現するためのものであり，2 次記憶装置で実現される．しかし，これは低速であるため，その記憶領域中で今後アクセスされる可能性の高いデータだけが主記憶に格納され，逆にアクセスされる可能性の低いデータは主記憶から仮想記憶へと退避される．

このようにすることで，論理的にはほぼ無限大の記憶空間を実現でき，また主記憶アクセスはほとんどの場合においてキャッシュ参照により高速に行えることとなるため，記憶装置の高速性と大容量性が両立できる．以下の節では，キャッシュ及び仮想記憶のそれぞれについて，その基本的な構成方式を概説する．

■6 群 - 4 編 - 4 章

4-3 キャッシュ

（執筆者：津邑公暁）[2011 年 8 月 受領]

4-3-1 キャッシュの基本構成

前述した空間的局所性に基づき，キャッシュには主記憶からある一定の大きさを単位としてデータがコピーされる．これをキャッシュブロックと呼ぶ．各キャッシュブロックは，それが記憶空間のどのアドレスのデータを保持しているかを記憶するためのタグ（Tag）を持つ．

ただし，タグでは，アドレスを構成するすべてのビットを保持する必要はない．例えば，キャッシュブロックのサイズが 64 バイトである場合，ブロックの先頭アドレスの下位 6 ビットは必ずすべて 0 となるため，その下位 6 ビットを除く上位ビットのみを記憶しておけばよい．

4-3-2 マッピング方式

主記憶へのアクセス要求があると，まずその要求されたアドレスを含むブロックのコピーがキャッシュ上に存在しているかどうかの検索が必要となる．この際，検索の複雑さは，キャッシュブロックと主記憶上の領域とのマッピング（対応づけ）をどう規定するかに左右される．

(1) ダイレクトマップ

ダイレクトマップ（Direct Map）は，主記憶アドレスから，そのデータを格納可能なキャッシュのエントリー（キャッシュライン）を一意に決定する方式である．ライン数 256，ブロックサイズ 64 バイトの例を図 4･1 に示す．

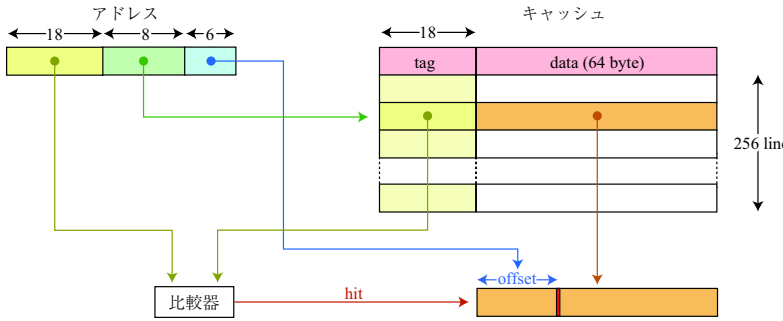

図 4･1　ダイレクトマップキャッシュの例

アクセスされるアドレスは 3 つの部分に分割され，中位アドレスはキャッシュラインインデクス（ライン数 256 のため 8 ビット）として，下位アドレスはキャッシュブロック内の相対アドレスオフセット（64 バイトのため 6 ビット）として使用される．また，上位アドレスはキャッシュタグとして記憶される．この場合，アクセスしたいアドレスの内容がキャッシュに格納されているかどうかは，以下のようにして判断できる．まず，中位アドレスに対応するキャッシュラインにアクセスし，そのラインに記憶されているブロックのタグと上位アドレ

スを比較する．一致すれば当該ブロックのデータ部に目的のデータが存在しており（キャッシュヒット），下位アドレスをオフセットとして当該アドレスにアクセスする．逆に一致しなかった場合，キャッシュ上には存在しない（キャッシュミス）．

ダイレクトマップでは，このように検索が一度の比較で実現できるため，構造が単純でハードウェア量が少なくて済むが，欠点もある．主記憶空間内には，中位アドレスは共通するが上位アドレスは異なるような領域が複数存在するが，これら複数の領域は単一のキャッシュラインしか割り当てられないため，同時にキャッシュ上に存在することができない．特にこれらが交互にアクセスされるような場合，お互いがお互いをキャッシュから追い出してしまうスラッシング（Thrashing）と呼ばれる状態となり，キャッシュヒット率が低下してしまう．

(2) フルアソシアティブ

対照的にフルアソシアティブ（Full Associative）は，主記憶アドレスとキャッシュラインの対応づけを全く行わない方式である．主記憶領域は任意のキャッシュラインに格納されることができるため，キャッシュヒット率は向上する．

しかし，検索時には主記憶アドレスとすべてのキャッシュラインのタグとを比較する必要がある．これを高速に実現するためには，連想メモリ（CAM：Content Addressable Memory）を用いる必要があり，消費電力やチップ面積の観点から，ライン数が多い場合に採用するのは困難である．

(3) セットアソシアティブ

上述した 2 つの手法の中間的な手法に，セットアソシアティブ（Set Associative）がある．これは，ダイレクトマップ同様，主記憶アドレスからキャッシュラインを一意に決定するが，各キャッシュラインに複数のキャッシュブロックを格納可能とする手法である．1 キャッシュラインに n 個のキャッシュブロックを格納可能なものを n ウェイセットアソシアティブと呼び，n をウェイ数あるいは連想度（Associativity）と呼ぶ．

検索時には，主記憶アドレスから決定されたキャッシュライン内に格納されている n 個のブロックが持つタグのそれぞれと上位アドレスとを比較する必要がある．

ここで，前述のふたつの手法は，セットアソシアティブにおける連想度 n が特殊な値をとる場合であることが分かる．すなわち，セットアソシアティブにおいて連想度 $n = 1$ としたものがダイレクトマップであり，連想度 n を最大（キャッシュ中の総ブロック数）としたものがフルアソシアティブであると考えることができる．

4-3-3 キャッシュブロックの置き換え

ダイレクトマップの場合，主記憶領域をキャッシュにコピーする際，そのコピー先ブロックは一意に定まるため，当該ブロックを使用していた領域がキャッシュから追い出される．しかし，セットアソシアティブでは当該キャッシュラインの n 個のブロックのいずれを追い出すかを．またフルアソシアティブの場合は，キャッシュ中の全ブロック中のいずれを追い出すかを，それぞれ選択する必要がある．

主記憶アクセスには時間的局所性があるため，最も適切と考えられるのは，アクセスした時刻が最も古いものを追い出すアルゴリズムである．これを LRU（Least Recently Used）と呼ぶ．しかし，LRU を実現するためには連想度 n 個のブロックのアクセス順をすべて記憶しておく必要があり，このためのハードウェアコストが大きくなるため，擬似的な LRU や，

ほかのより単純なアルゴリズムが用いられる場合もある．

4-3-4 主記憶更新方式

記憶空間に対しては当然読出しだけではなく書込みも行われるが，キャッシュに対する書込みが発生するとキャッシュと主記憶間で矛盾が生じる．このため，キャッシュ上の変更を主記憶に反映する必要がある．

(1) ストアスルー

ストアスルー（Store Through）またはライトスルー（Write Through）は，キャッシュへの書き込みが発生した際に同じ更新を主記憶に対しても行う方式である．変更は主記憶に即座に反映され制御も比較的単純で済むが，毎回主記憶書き込みが発生するため，ライトバッファを備えるなどの対策をしなければ，主記憶アクセス時間により性能が低下してしまう．

(2) ライトバック

一方，書き込み発生時にはキャッシュのみを更新し，当該キャッシュブロックを更新した（すなわち主記憶と整合性がとれていない）ことを記憶しておく方式が，ライトバック（Write Back）である．そして，当該ブロックがキャッシュから追い出される際，その内容を主記憶に書き戻す．この方式では，同一ブロックに対して複数の書き込みがあった際にも主記憶への反映は一度で行えるため，主記憶への書き込みアクセスを最小限に抑えることができる．

4-3-5 ユニット分割

キャッシュは，複数ユニットに分割して構成することによって，より性能を向上させることができる場合がある．ここでは，その幾つかの方式を紹介する．

(1) マルチレベルキャッシュ

メモリ性能が CPU に対して相対的に低下している状況を受け，記憶階層の構成要素であるキャッシュ自体を更に多段構造にすることが一般的となっている．2 レベルの場合，CPU に近く，より高速・小容量なキャッシュを 1 次キャッシュ（Primary Cache）と呼び，他方を 2 次キャッシュ（Secondary Cache）と呼ぶ．

(2) ハーバードアーキテクチャ

主記憶上に保持される情報としては命令及びデータがあるが，これらに対するアクセスの特徴は異なる．命令に対しては基本的に読み出しのみが行われ書き込みが発生しないが，データはそうではない．また，命令パイプライン（7 章）では命令フェッチとデータフェッチで競合が発生する場合がある．よって，命令用およびデータ用でキャッシュメモリを分割し，並列にアクセス可能とする方法が有効となる．このような構造はハーバードアーキテクチャ（Harvard Architecture）と呼ばれる．

(3) ヴィクティムキャッシュ

ヒット率低下を防ぐ方法の一つはウェイ数を増大させることであるが，すべてのラインに対しタグ比較回数が増大してしまう．一方で，競合の発生頻度にはラインごとにばらつきがある．このような場合，ヴィクティムキャッシュ（Victim Cache）が有効である．

元来ヴィクティムキャッシュは，追い出されたブロックを保持しておくための，ごく小容量なフルアソシアティブキャッシュとして考案された．頻繁に競合が発生するラインのみ連想度を向上させたような効果が得られ，全体のウェイ数を増大させずにヒット率向上が見込める．

最近では，ある程度の大きさを持つ主記憶直前のキャッシュ（LLC：Last Level Cache）を，前段から追い出されたデータを保持するヴィクティムキャッシュとして構成する例もある．

■6 群 - 4 編 - 4 章

4-4 仮想記憶

（執筆者：津邑公暁）［2011 年 8 月 受領］

4-4-1 仮想記憶の必要性

主記憶素子が非常に高価であった時代，プログラムやデータを格納するための十分な記憶領域を確保できず，実記憶領域よりも大きい仮想的な記憶空間を提供することが必要であった．また，このような問題がなくなった現在でも，多数のプロセスが同時に実行される状況が一般的となったことから，各プロセスが固有の記憶空間が使用できることが望ましい．

そこで，記憶空間を仮想化し，仮想記憶と物理記憶をマッピングさせる方法がとられる．仮想記憶のアドレス空間は磁気ディスクなどの 2 次記憶装置に格納され，そのうち必要なブロックが主記憶にコピーされて，アクセスされる．この際，仮想アドレスから物理アドレスへの変換は，メモリ管理ユニット（MMU：Memory Management Unit）によって行われる．

本節では，セグメント方式及びページングと呼ばれる 2 つの方式について述べる．

4-4-2 セグメント方式

セグメントと呼ばれる可変長の仮想記憶空間を単位とし，これを各プロセス等に固有に割り当てることで，他プロセスの存在を意識することなく記憶空間を利用可能となる．これをセグメント方式（Segmentation）と呼ぶ（**図 4･2**）．

セグメントは物理空間上の連続領域にマッピングされる．このため，仮想空間内のアドレスは，その空間がマッピングされた物理領域の先頭アドレス（ベースアドレス）に対するオフセットとなり，これらを加算することで物理アドレスに変換できる．このため，各セグメントとそのベースアドレスとの対応を，セグメントテーブルと呼ばれる表で記憶し，管理する．

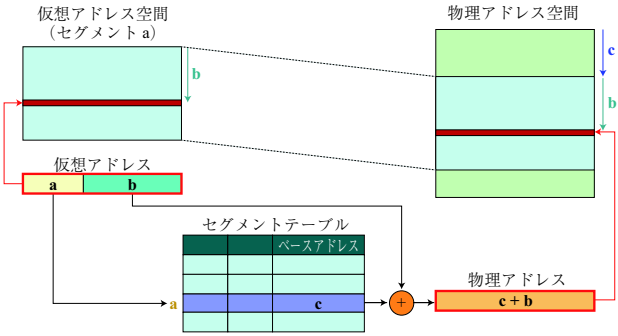

図 4･2　セグメント方式のアドレス変換

しかし，セグメントは可変長であるため，プロセスの開始・終了が繰り返されると，記憶領域中に小さな空き領域が散在する状態になり，総空き容量が十分であっても，十分な大きさの連続領域が確保できなくなるという欠点がある．この状態をフラグメンテーション（Fragmentation）と呼ぶ．

4-4-3 ページング

フラグメンテーションを避けるためには，主記憶を固定長単位で管理することが有効である．ページング（Paging）は，仮想空間をページと呼ぶ固定長の領域に分割し，これを単位とする方式である．物理空間も同じ固定長単位のページフレーム（Page Frame）と呼ぶ領域に分割し，ページをこのページフレームにマッピングする．

ページサイズは一般に 2^n バイトと設定される．例えば 4K バイトの場合，各ページの先頭アドレスの下位 12 ビットは必ず 0 となる．ページフレームもまた同じである．よって，この下位アドレスがページ内の相対アドレス（オフセット）となる．また，上位アドレスをページやページフレームの番号（ID）とすると，ページ番号を表す上位ビットを対応するページフレーム番号に置換するだけで，物理アドレスへの変換が可能となり，セグメント方式のような加算は必要ない．この，現在各ページがどのページフレームにマップされているかを記憶する対応表をページテーブルと呼ぶ．

ただし，このページテーブルは記憶空間上に置かれるため，幾つかの解決すべき問題がある．一つは，アドレス変換のためのページテーブルへのアクセスにより，主記憶アクセス回数が倍増してしまうことである．そこで，最近変換したページ番号とページフレーム番号の対応を記憶しておくための，TLB（Translation Lookaside Buffer）と呼ばれるキャッシュを設けることでこれを緩和する．もう一つは，仮想アドレス空間が巨大である場合エントリー数が膨大となり，ページテーブル自体が主記憶を圧迫してしまうことである．そこで，ページテーブルを多段化することでこれを解決する．2 段階化した場合の例を**図 4･3** に示す．

ページ集合の ID を表す上位アドレスで 1 段目のテーブルを引き，そのページ集合に対応する 2 段目テーブルのアドレスを得る．次に中位アドレスを用いてこの 2 段目テーブルを引くことでページフレーム番号が得られ，オフセットである下位アドレスと合わせて物理アドレスを得る．従来のページテーブルは複数の 2 段目テーブルに分割された形となるが，これらはすべて保持しておく必要がなく，テーブル自体を仮想記憶に追い出したり，必要になるまで作成しないといった方法をとることで，実記憶領域を有効に利用可能となる．

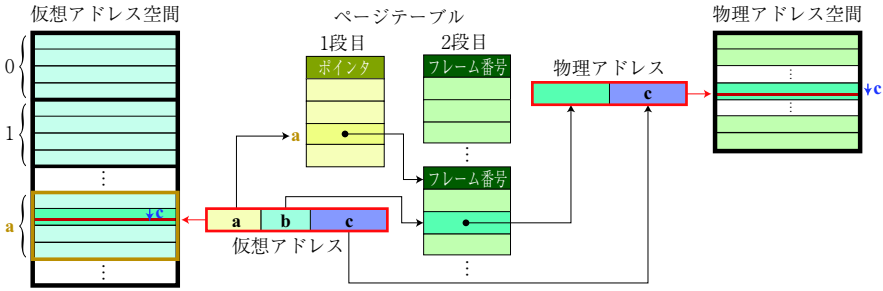

図 4･3　多段ページングのアドレス変換

なお，前述したセグメントをページ単位で構成すれば，セグメントと同じ大きさの連続領域は必要なくなり，フラグメンテーションの問題も解決する．これをページ化セグメンテーションと呼ぶ．ページテーブルはセグメントごとに必要となるが，多段ページングの場合と同様に，アクセスがないセグメントに対応するページテーブルは保持しておく必要がない．