第１１回　キャッシュのマッピング方式・不一致・仮想記憶　　（第６章　キャッシュメモリ）

［１］　マッピング方式と追出しの効果

［補足］ マッピングは、本来のアドレス（主記憶のアドレス）と、それを納めるキャッシュの場所との対応関係の付け方である。CPUが主記憶のXXXX番地のデータが欲しいと言った時、それをキャッシュに置くのにどこへ置くのか、ということである。教科書 p80 にあるように3つの方法がある。

(1) のダイレクトマッピングは、主記憶アドレスの一部をキャッシュのブロック番号にする方法で、たとえばブロックの大きさを 64 バイト（アドレス下位6ビット分）、ブロック数を 512 個（64 バイト×512 個＝32K バイト、アドレスのビット位置は下から15ビット目から7ビット目までの9ビット分）を充てることにする （絵を描いてみよ）。 このとき、アドレスの下から16ビット目より上位のビットのパターンが違っても、キャッシュ内の同じブロックへ割当てられることになる。 ≪教科書図 8.3 の例では主記憶上の連続するブロック a, b, c がキャッシュ内の同じブロックへ割当てられている例であるが、今ここに書いたやり方だと主記憶上の 512 個おきのブロックがキャッシュ内の同じブロックへ割当てられる。≫

(2) のフルアソシアティブマッピングは、主記憶上のアドレスに関係なく、キャッシュ上のどのブロックに置くこともできる。この場合、CPUから出されたアドレスからキャッシュ内のブロック位置を探すためには、表を引かなければならない。表を引くのは（ダイレクトマッピングがアドレスの一部ビットを切り出せばよかったのに比べて）ハードが複雑になる（表のためのメモリが欲しくなる）。

(3) のセットアソシアティブマッピングは、(1)と(2)の折衷案である。キャッシュ内をグループ化しておき、(1)で変換した後ぶつかっても複数ブロックを入れられるようにしておき、その同じグループ内のブロックは(2)の方法で検索する。たとえば Intel の I7 2 で≪8-way set associative≫というのは、1つのグループが8つのブロックを入れられるようになっている。

Patterson & Hennesy ″Computer Organization and Design Revised Fourth Edition″ の Figure 5.39 (p.541) によると、Intel の Nehalem （i7 の第1世代）では、

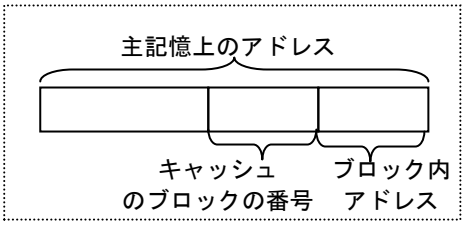
L1（レベル1）キャッシュ： 命令とデータは別で、それぞれ 32KB、命令キャッシュは 4way, データキャッシュは 8way の set associative、ブロック長は 64 バイト、write-back/write-allocate、

L2（レベル2）キャッシュ： コアごとに1つ、命令とデータは一緒 (unified)、256KB、8way set associative、ブロック長は 64 バイト、write-back/write-associate、

L3（レベル3）キャッシュ： チップ全体に1つ（コアに共通）、命令とデータは一緒 (unified)、16way set associative、8MB、置換え手法は not available、ブロック長 64 バイト、write-back/write-allocate。

a.　ダイレクトアドレスマッピング方式について、次の問に答えよ

CPU が主記憶中の 12345678_{16} 番地～$1234567B_{16}$ 番地の4バイトからなる命令を読み出したいとする。アドレスマッピングの方式として、上記[補足]にあるような(1)のダイレクトマッピング方式（ブロックサイズ 64 バイト、総ブロック数 512 個）とするとき、12345678_{16}～$1234567B_{16}$ 番地はキャッシュ中のどの部分（何番目のブロックの中の、何バイト目）に格納されるか？



b.　同じ主記憶アドレスへのアクセスに対して、アドレスマッピング方式として(2)のフルアソシアティブ方式を用いる場合、キャッシュ中のどの部分（何番目のブロックの中の、何バイト目）に格納されるか

c.　上記aやbのように、主記憶の 1234 5678_{16} 番地を含むブロックが、キャッシュに取り込まれたとしよう。次に、主記憶の 2345 5644_{16} 番地がアクセスされたとする。ダイレクトマッピング方式では 2345 5644_{16} 番地に対してキャッシュ上のどのブロック番号が使われるか？　そのときに何が起こるか？　また、フルアソシアティブ方式では、どのブロック番号が使われるか？　そのときに何が起こるか？

d.　キャッシュのマッピング方式について、間違っているものを選べ

① ダイレクトマッピング方式は、キャッシュ内のブロック位置を求めるのが容易で計算が速いが、フルアソシアティブ方式は、表の検索を行う必要があるので求めるのが複雑になる。

② フルアソシアティブ方式では、主記憶のアドレスからキャッシュ内のブロック位置を求めるのに、対応の表を用いるが、表の中で主記憶アドレスが合致する行を求めるために検索する必要があり、特殊な「連想メモリ（p87 に簡単な説明あり）」を用いる。

③ フルアソシアティブ方式では、キャッシュ内のブロック位置がアドレスから一意に決まってしまうので、既にその位置が使われている場合には（衝突した場合）すぐに上書きされてしまう。

④ ダイレクトマッピング方式では、キャッシュ内のブロック位置がアドレスから一意に決まってしまうので、既にその位置が使われている場合には（衝突した場合）すぐに上書きされてしまう。

e.　教科書 p80 下から4行目からの、ブロックの追い出しの方法の説明について、正しいものを選べ

① キャッシュ内ブロックの追い出しの候補を選択する必要があるのは、ダイレクトマッピング・フルアソシアティブとセットアソシアティブ方式のすべてである

② キャッシュ内ブロックの追い出しの候補を決める方法は、最も古く到着したものから先に追い出すFIFO方式が効率がよい

③ キャッシュ内ブロックの追い出しの候補を決める方法は、最近最も使われていないもの（最後に使われた時点が最も古いもの）から先に追い出すLRU方式が効率がよい

④ ブロックの追い出しの候補を決める方法は、ランダムに選ぶ方法がもっとも効率がよい

［２］　書込みと不一致問題

［補足説明］

キャッシュは主記憶のコピーを高速にCPUに提供する。キャッシュの内容は、対応する主記憶の内容と同じでなければならない。ところが、メモリへの書込み動作をすると、主記憶へ書込んだつもりが、キャッシュまでしか書込まれておらず、主記憶がアップデートされていない、という事態が起こりえる。この状況をキャッシュと主記憶の間で「不一致」がある、と言う。

a.　キャッシュメモリの、書込み操作時の動作について、下記の動作項目のそれぞれに順番を示す番号を書け

①（　　） キャッシュは書込み要求を受け取って、キャッシュ内に当該主記憶アドレスのブロックが存在するかチェックする

②（　　） 書込みで変更されたキャッシュブロック内容を、主記憶へ書き戻す

③（　　） もしキャッシュ内にブロックがあれば、キャッシュブロック内の該当するアドレスに書込みを行う.　もしなければ主記憶からキャッシュ内へブロックを転送し、その上でキャッシュブロック内の該当するアドレスへ書込みを行う

④（　　） CPU から、（主記憶アドレス、書込みデータ）を添えて、書込み要求がキャッシュに対して送出される

（注：　③の主記憶への下記戻しのタイミングは、次で議論する。）

b.　キャッシュの書込み時の動作について、教 p.81 の 8.1.3 を参考にして、次の説明の空欄を埋めよ。

キャッシュの書込み時の動作には、ライトスルーとライトバックの2つのタイプがある。ライトスルーは、(①)への書込みと同時に(②)への書込みを行う。ライトバックは、(③)への書込みがあっても、直ぐには(④)への書込みを行わず、キャッシュブロックが(⑤)ときに(⑥)へ書込む。

両者を比較すると、ライトスルーは(⑦)書込みのたびに(⑧)へも書込みその完了を待つが、ライトバックは(⑦)書込みのたびに(⑧)へ書込むというわけではなく殆どの場合は(⑦)だけ書込む。 従って、ライトスルーは書込み時のメモリアクセス時間は(⑨)と同等になるが、ライトバックは書込み時のアクセス時間は(⑩)と同等になり、(⑪)くなる。

他方、キャッシュ上の内容を比較すると、ライトスルーでは(⑫)と(⑬)の内容は常に⑭(一致する／一致しない)のに対し、ライトバックでは⑮(一致する／一致しない)ことが多くなる。つまりライトバック方式はキャッシュ内容の(⑯)問題を抱えることになる。

また、ライトバックは制御も複雑になるが、書込み時のアクセス時間も読出し時と同様に短縮したいという要求があるので、実際には広く用いられている。

c.　キャッシュメモリの不一致問題について、正しいものを選べ

① キャッシュメモリの不一致問題とは、キャッシュメモリの内容と主記憶の内容が不一致になる問題であり、主記憶に直接書き込み操作が行われたときに発生する

② キャッシュメモリの不一致問題とは、キャッシュメモリの内容と主記憶の内容が不一致になる問題であり、キャッシュメモリに書き込み操作が行われたがそれを主記憶に反映しないときに発生する

③ キャッシュメモリの不一致問題とは、キャッシュメモリ内の複数のブロックにたまたま同じ内容があった場合、それらの間の不一致が問題になる

④ キャッシュメモリの不一致問題とは、1つのCPUのキャッシュメモリの内容と別のCPUのキャッシュメモリの内容が不一致になる問題であり、主記憶に直接書き込み操作が行われたときに発生する

d.　CPUからの書き込みアクセスの手法である、ライトスルーとライトバック方式について、正しいものを選べ

① ライトスルーはCPUからの書込みを、キャッシュには書込まずスルーし、主記憶だけに書込む。同じブロックを次回読出すときに、主記憶からキャッシュへコピーを転送するので、その時点でキャッシュに変更が反映される。ライトバックはCPUからの書込みの度ごとに主記憶に書込むのではなく、書込みによる変更はキャッシュにとどめておき、そのブロックをキャッシュから追出すときに主記憶に書き戻す。

② ライトスルーはCPUからの書込みを、キャッシュに書込むと同時に主記憶にも書込む。書込みに要する時間は、主記憶の書込みが完了するまで待たなければならないので、キャッシュを使っていない状況と同じになってしまう。ライトバックはCPUからの書込みの度ごとに主記憶に書込むのではなく、書込みによる変更はキャッシュにとどめておき、そのブロックをキャッシュから追出すときに主記憶に書き戻す。

③ ライトスルーはCPUからの書込みを、キャッシュには書込まずスルーし、主記憶だけに書込む。同じブロックを次回読出すときに、主記憶からキャッシュへコピーを転送するので、その時点でキャッシュに変更が反映される。ライトバックはCPUからの書込みの度ごとに主記憶に書込むのではなく、書込みによる変更はキャッシュにとどめておき、そのブロックに対する次の読込みのときにキャッシュから主記憶にコピーして統一をとる。

④ ライトスルーはCPUからの書込みを、キャッシュに書込むと同時に主記憶にも書込む。書込みに要する時間は、主記憶の書込みが完了するまで待たなければならないので、キャッシュを使っていない状況と同じになってしまう。ライトバックはCPUからの書込みの度ごとに主記憶に書込むのではなく、書込みによる変更はキャッシュにとどめておき、そのブロックに対する次の読込みのときにキャッシュから主記憶にコピーして統一をとる。

［３］　仮想メモリ

仮想メモリはオペレーティングシステムの講義で取上げられるので、詳細は必要ないかもしれない。ただ、キャッシュメモリと

まったく同様に、記憶の階層の考え方に従っており、またブロックの選択、置換えアルゴリズムなど、同様の問題を議論するので、ここで参照しておく。仮想メモリとキャッシュメモリの最大の違いは、キャッシュメモリは1命令の実行中に起こる高速の動作であり、仮想記憶はそれに比べるとかなり緩慢な動作である。従って、キャッシュメモリでは仕掛けを原則としてハードウェア化して高速化を図るが、そのために余り複雑なアルゴリズムは取り込めない。それに対して、仮想記憶では、通常の（高速側のメモリにあるときの）アクセスはハードウェアで制御するが、ブロックの置き換え等は多少遅くてもよい設計としておいてソフトウェアで実現し、そのためにそれなりに複雑なアルゴリズムを利用できるところが、相違点になる。また、高速側のメモリの大きさの違い（キャッシュの場合キャッシュ容量は数百キロバイト～数メガバイト、仮想メモリの場合は高速側は主記憶を使うが主記憶容量は数百メガバイト～数ギガバイト）、ブロックの大きさの違い（キャッシュの場合ブロックは数十～数百バイト、仮想メモリの場合ブロックは 4096 バイトが普通）、アドレス変換表の大きさの違い（キャッシュの場合は数千エントリー程度でセットアソシアティブ方式でハードウェアによる連想検索を利用、仮想メモリの場合は1メガエントリー以上になることもありフルアソシアティブ方式だが要求アドレスをアドレス変換表のオフセットとして高速アクセス）などの違いがある。

もう1度教科書 p65 の図 7.3「メモリ装置の階層」を見て欲しい。 キャッシュメモリはCPUと主記憶の間にあって、（レジスタはちょっと脇においておく、）CPUから主記憶へのアクセスを（見かけ上）高速化するものであった。 それに対して仮想メモリは、CPUが主記憶をアクセスするときに、裏にある補助記憶（ハードディスクなど）に実体を置き、「キャッシュメモリー主記憶」の関係をそっくり「主記憶ーディスク」の間に同じように作ったものである。 つまり、キャッシュのときは「主記憶上のデータの一部を高速なキャッシュに置き、局所性の原理によって大抵のアクセスがキャッシュで済むので、有効アクセス時間がキャッシュに近づく」というシナリオだったが、仮想メモリでは「ディスク上のデータの一部を高速な主記憶に置き、局所性の原理によって大抵のアクセスが主記憶で済むので、有効アクセス時間が主記憶に近づく」というシナリオになる。
では、何のために仮想メモリを使うのか？ もし主記憶が十分に大きければ（最近の主記憶は実際かなり大きい）、すべてを主記憶に置けばそれで済むはずである。（キャッシュメモリの場合は、キャッシュメモリ自体が小さいので、全てのデータをキャッシュメモリに置くわけには行かなかった。） 仮想メモリを使う理由は、①やはり主記憶だけでは容量が不十分である、②別に「アドレスの読替え」をしたい理由がある、の2点であろう。 第1点目の容量の問題は、まあ仮に現在のPCの主記憶容量が2Gバイトであっても、いろいろな（アプリケーション）プログラムのメモリ必要量はどんどん増えているし、もう1つは、OSの授業で学ぶが、複数の仕事を同時に実行する（CPUを時間分割で切替えて使っている）ことが多いのでそれらのプログラムを全て主記憶に置くとなると、主記憶の容量が足りなくなる。 だから、主記憶の見かけ上の容量を増やすために、仮想メモリを使う。第2点目のアドレス読替えは、第1点目で触れた≪複数のプログラムを同時に走らせる≫場合、それぞれのプログラムで使うオペランドのアドレスは、先頭が0番地であることを想定してコンパイルされるが、同時に走らせる2つ目のプログラムは1つ目のプログラムより後のアドレスを使わざるを得ない。たとえば2つ目のプログラムが 10000 番地から始まる領域を使うとすると、プログラム中の命令のオペランドアドレス指定はすべて 10000 番地分底上げしてやる必要が出てくる。（これをリロケーションと呼ぶ。オペレーティングシステム（OS）の授業で触れられるだろう。） このため、アドレスの読替えができると具合がよい。

［補足］ 教科書 p83 の 8.2.2 分割方式については、MS Windows や Linux では、p85 の 3 行目に書いてある「ページセグメンテーション方式」が使われている。このあたりの細かいことは、オペレーティングシステム（OS）の授業で議論することになるだろう。

a. 仮想メモリの仕組みについて、正しいものを選べ
   ① 仮想メモリは、補助記憶装置（一般的にはハードディスク装置）に置かれた情報を、主記憶（メインメモリ）に持ってきてCPUからアクセスする。
   ② 仮想メモリは、主記憶に置かれた情報を、補助記憶装置に持ってきてCPUからアクセスする。
   ③ 仮想メモリでは、補助記憶装置上に仮想メモリ空間を置き、CPUは仮想メモリ空間にある情報を直接アクセスす

る。

④ 仮想メモリでは、主記憶の容量に比べて仮想メモリ空間の大きさを大きく出来るので、アクセス速度は（仮想メモリを使わない場合に比べて）速くなる

b. 仮想メモリの仕組みについて、誤っているものを選べ

① 仮想メモリは、ハードディスク上に置いた仮想メモリ空間の一部を、主記憶上にコピーし、CPUは主記憶から情報を読んだり書いたりする。

② 仮想メモリでは、CPUは仮想メモリ空間に対するアドレス（＝仮想アドレス）を出してアクセスしようとし、それを主記憶上のアドレスに変換して、主記憶に実際にアクセスする。

③ 仮想メモリは、主記憶の容量に比べて仮想メモリ空間の大きさを大きく出来るので、主記憶より大きなプログラムを置いたり、複数の大きなプログラムを置いて主記憶容量を超えるようになっても、使うことが出来る。

④ 仮想メモリでは、CPUからのアクセスが最終的に仮想メモリ空間のあるハードディスクへのアクセスになるので、アクセス時間はハードディスクのアクセス時間になる。

c. ページについて、正しいものを選べ

① ページは、主記憶を都合のよい可変長で区切ったもので、その長さはソフトウェア（オペレーティングシステム）が管理している

② ページは、主記憶を一定の固定長で区切ったもので、その長さはソフトウェア（オペレーティングシステム）が管理している

③ ページは、主記憶を一定の固定長で区切ったもので、その長さはCPUのハードウェアによって決まっている

④ ページは、主記憶を一定の固定長で区切ったもので、その長さはハードディスクのブロック長（セクター長）によって決まっている

d. ページについて、誤っているものを選べ

① 実アドレス空間（主記憶）上のページの大きさと、仮想アドレス空間（ハードディスク）上のページの大きさは、同じでなければならない

② 実アドレス空間上のページの番号（空間内のページの位置）の並び方は、仮想アドレス空間上のページの番号（空間内の位置）の並び方と同じでなければならない

③ CPUからのアクセスに伴って出される仮想アドレスは、仮想ページ番号とページ内オフセットに分解されるが、その内で仮想ページ番号のみが実アドレス空間上のページ番号に変換され、ページ内オフセットは変換されずにそのまま使われる。

④ 仮想ページ番号、実ページ番号、ページ内オフセットは、いずれもハードウェアで決まっている。

e. 仮想メモリのアドレス変換について、正しいものを選べ

① 仮想メモリのアドレス変換は、キャッシュメモリのアドレス変換と同じように、セットアソシアティブ方式が主に使われている

② 仮想メモリのアドレス変換は、簡単化のために、ダイレクトマッピング方式が使われている

③ 仮想メモリのアドレス変換は、柔軟性を得るために、フルアソシアティブ方式が使われている

④ 仮想メモリのアドレス変換は、偏りの少ないハッシュ関数を用いた計算による方式が使われている

f. 仮想メモリのアドレス変換に使われるページテーブルについて、間違っているものを選べ

① ページテーブルは、通常は仮想ページ番号を行位置番号とし、行の内容として実ページ番号を入れておく。これによって仮想アドレスから実アドレスを比較的高速に求めることが出来る。 （図 8.12 参照）

② ページテーブルの行数（エントリー数）は、仮想ページ番号を行位置番号とすると、仮想ページの数だけ必要になり大きくなるという問題がある。

③ ページテーブルの検索は、与えられた仮想ページ番号と表の中の仮想ページ番号欄とを上から比較してゆき、一致した行の実ページ番号を取り出す。

④ ページテーブルの参照を高速化するために、ハードウェアで高速に参照できるTLB（Table Lookaside Buffer）が置

かれている。 TLBは連想メモリのハードウェアを使い、仮想ページ番号が一致する行を高速に（1 サイクルで）見つけることができる。

［補足］ ページテーブルのやっかいな点は、エントリー数が（仮想アドレス空間が大きくなるにつれて）どんどん増えてきていることである。ページテーブルはなるべく実メモリ上に置きたいため、教科書 p87 にあるように2レベルのマッピング（3レベルもある）とし、図 8.13 のレベル2テーブルは現在使うものだけを中心に実メモリ上に置くような工夫をしている。
≪仮想メモリについては、ここまでにする。これ以上細かい話、たとえば追出しアルゴリズムの比較などは2年のオペレーティングシステムの講義に譲る。≫

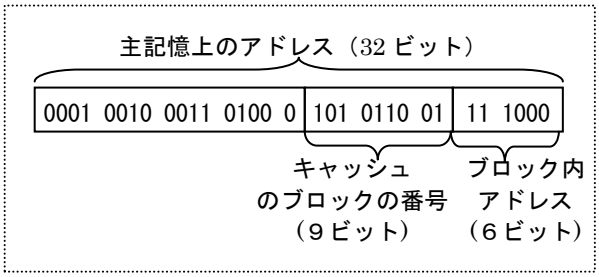
≪解答≫

[1]

a.　下記の主記憶アドレス

1234_{16} ＝ 0001 0010 0011 0100 0101 0110 0111 1000_2

を、17＋9＋6ビットに分割して（中位9ビットをブロック番号として、下位6ビットをブロック内のバイトアドレスとして）用いるので、右図のようになる。



キャッシュのブロック番号 ＝ 101011001 ＝ 159_{16}

ブロック内のバイトアドレス ＝ 111000 ＝ 38_{16} から4バイト分

つまり、38・39・3A・$3B_{16}$ の4バイト分である

どのブロックに格納されるかは、このブロック番号 159 によって決まるのであり、上位の17ビット（0001 0010 0011 0100 0）には関わらない。つまり、上位の17ビットが何の値を取ろうと、中位9ビットのパターンが 101 0110 01 であればブロック番号 159 のブロックに書き込まれるのである。 これが「ダイレクトマッピング方式」の原理である。

b.　フルアソシアティブ方式では、ブロック番号は主記憶アドレスから決まるのではなく、「空いているブロック」を利用する。未だキャッシュ上にこの主記憶アドレスに対応するブロックが存在しないときには、たとえば空いているキャッシュ上のブロック 78 番が見つかったとすれば、そのブロック 78 番へ主記憶の 1234 5640～1234 $567F_{16}$ の内容のコピーを作る、というようにする。

（注： キャッシュが満杯になれば、「空いている」ブロックは存在しないので、既に使っているブロックを(主記憶へ)追出して、空きを作ることになる。誰を追出すかを決めるやり方を、「キャッシュの追出しアルゴリズム」と呼ぶ。）

次に主記憶の 1234 5678 番地 （もしくは、1234 5640～1234 567F 番地のいずれか）にアクセスするときは、既にキャッシュにコピーがあるのだから、それを使えばよいのだが、さてそれをどうやって見つけるのか？ ダイレクトマッピング方式なら中位9ビットのアドレスによってキャッシュ内のブロック番号が一意に決まるから、そこに欲しい 1234 5678 番地が入っているかどうかを確かめれば良かったが、フルアソシアティブ方式では、見つけるためには、すべてのキャッシュブロックについて対応する主記憶アドレスがどれであるかを見て 1234 5678 （正確にはブロック先頭番地の 1234 5640） を検索しなければならない。 つまり、ここに「検索」の作業が必要になり、これには（キャッシュのアクセス時間に比較して十分重荷になるだけの長い）時間がかかる。

c.　ダイレクトマッピング方式では、主記憶の 2345 5644_{16} ＝ 0010 0011 0100 0101 0101 0110 0100 0100_2 番地は、中位の9ビットの部分 101 0110 01_2 ＝ 159_{16} で決まるブロックに格納される。つまり使用するブロック番号は 159_{16} であり、1－1でアクセスしたブロックと同じブロックである。従って、1－1でアクセスした主記憶の情報はもはやキャッシュ内に留め置くことはできず、捨てざるを得ない（新しくアクセスした 2345 5644 を含むのブロックの内容で、上書きして消してしまうことになる）。これは、たとえ他のブロックが使われておらず空いていたとしても、（アドレスの中位9ビットで指定されるのだから）捨てざるを得ない。

他方、フルアソシアティブ方式では、どのキャッシュブロックを使うかは、ブロックの空き具合による。たとえば1－2のアクセスで、主記憶の 1234 5678 番地を含むブロックをキャッシュ上のブロック 78 番へ置いたとする。もしキャッシュ上のブロック 5 番が空いていれば、78 番はそのままにして、主記憶 2345 5644 番地を含むブロックを 5 番に入れることができる。別の言葉で言えば「キャッシュ上のブロックは衝突しないで済んだ」。

d.　③　フルアソシアティブ方式ではキャッシュ内のブロック位置(ブロック番号)は、(任意の)空いている場所を使うことができる。アドレスから一意に決まってしまう方式は、ダイレクトマッピング方式である。

e.　③

①は、ダイレクトマッピング方式は追い出し候補の選択をする必要がない。なぜなら、与えられた主記憶アドレスから、キャッシュ内のブロック番号が一意に決まるからである。

②③④は、追い出し方式の比較であるが、一般に③の LRU 方式が効率がよい（具体的には、後で必要なものをなるべく残すようにするので、ヒット率が高くなる）傾向がある。 ④のランダムに決める方式は「当るも八卦当らぬも八卦」ということになる。②の FIFO（先入れ先出し）方式は、必ずしも「新しいものほど今後も使われ続ける」ということが言えず、そこそこのヒット率に留まる。

［2］

a. ①（2） ②（4） ③（3） ④（1） ④→①→③→②の順

b. ①（キャッシュ） ②（主記憶） ③（キャッシュ） ④（主記憶） ⑤（追出される） ⑥（主記憶）
⑦（キャッシュ） ⑧（主記憶） ⑨（主記憶のアクセス時間） ⑩（キャッシュのアクセス時間） ⑪（速く）
⑫（キャッシュ） ⑬（主記憶） ⑭（一致する） ⑮（一致しない） ⑯（不一致）

c. ② （④の問題はマルチプロセッサシステムで発生する別の種類の不一致問題であり、解決には工夫を要する）

d. ② （③はキャッシュ自体には書き込みをしないと言っているが、これだと次に読出しアクセスをしたときに内容が正しくないことになる。 ④は「読込みのときに統一を取る」としている点が誤りで、このやり方の問題は次の読出しの時に主記憶から読み出すので時間がかかる点にある）

［3］

a. ①

②は、CPUから補助記憶装置（ハードディスクなど）にアクセスするように書いてあるが、CPUは命令読出しや命令の操作対象のデータ（オペランド）を、補助記憶から持ってくることは出来ない。

③は、「補助記憶装置上に仮想メモリ空間を置き」は正しいが、「CPUは仮想メモリ空間…直接アクセスする」は②と同じ理由で間違い

④は、「主記憶に比べて仮想メモリ空間の大きさを大きく出来る」は正しいし、実際にそのようにされているが、「アクセス速度は速くなる」は間違い。仮想記憶でもキャッシュのアクセス時間のモデルと同じように、hをヒット率とするとき、

実効アクセス時間 ＝ h×高速メモリ（主記憶）アクセス時間 ＋ （1－h）×低速メモリ（ディスク）アクセス時間

が成り立ち、高速メモリ（主記憶）だけの場合と比較すると、第2項の分だけ遅くなる

b. ④

①は正しい。キャッシュのときと同じように、ハードディスクから主記憶へ一定の大きさのブロック単位でコピーするが、このブロックのことを「ページ」と呼ぶ。

②は正しい。CPUは仮想アドレスを出しそれをアドレス変換ユニットで主記憶上のアドレス（物理アドレス）に変換する。

③は正しい。一般に物理アドレスのビット長より仮想アドレスのビット長のほうが長い。

④は間違い。実効的なアクセス時間は3－1の④の説明に書いてある通り。

c. ③

①②④は間違い。ページの大きさは固定長（システム上の全てのページの大きさは同じ）であり、しかも、CPUハードウェアで決まってしまう。そのため、同じCPUチップを使う限りOSの種類にも依らず同じになってしまう。汎用のCPUでは4Kバイト＝4096 バイトが多い。

d. ②

①は正しい。キャッシュでも同じだったが、高速メモリ側のブロック長と低速メモリ側のブロック長は同じ大きさである。なぜなら、それらの間で移動（コピー）するから。

②は間違い。実アドレス（物理アドレスとも言う。主記憶（＝高速メモリ側）上のアドレスのこと）は、仮想記憶では（キャッシュと違い）フルアソシアティブと同等の方式を取るので、必要に応じて空いているブロックを次々使うことになる。だから、アドレス順に並ぶことは無くて、乱雑に並ぶ。

③は正しい。キャッシュでもCPUから出されたアドレスは3つの部分に分かれ、上位と中位部分を併せた部分が仮想ページ番号に、下位部分（ブロック内オフセットアドレス）がページ内オフセットに対応すると考えられる。オフセット部分は変換されずにそのままブロック（ページ）内アドレスとして使われ、上位＋中位の部分が変換される。

④は正しい。アドレスの変換はハードウェアが支援しており、たとえばヒット時のアクセスはすべてがハードウェアで処理される。そのため、これらの部分のビット長はハードウェア（CPUチップの設計）で決まっている。

e. ③　仮想記憶のアドレス変換はフルアソシアティブと同等で、仮想アドレス⇒物理アドレスの対応表を使って変換する。

f. ③

①は正しい。 仮想ページ番号Pを与えると、表のP行目を読んで、実ページ番号（物理ページ番号）を得る。

②は正しい。 最近は仮想メモリの量が大きくなり、単純な表だと非常に大きな表になるので、様々な工夫がされている。

③は誤り。①にある通りである。③のようにすると、検索に時間がかかり、メモリのアクセス速度が低下して具合が悪い。

④は正しい。TLBは『ページテーブルの内容を一時的に「キャッシュ」しておく高速メモリ』という位置付けになる。

≪基本情報処理技術者試験問題から≫

1） キャッシュメモリに関する記述のうち，適切なものはどれか。　（基本 21 春 12）（基本 18 春 22）（基本 16 秋 21）

ア 書込み命令を実行したときに，キャッシュメモリと主記憶の両方を書き換える方式と，キャッシュメモリだけを書き換えておき，主記憶の書換えはキャッシュメモリから当該データが追い出されるときに行う方式とがある。

イ キャッシュメモリにヒットしない場合に割込みが生じ，プログラムによって主記憶からキャッシュメモリにデータが転送される。

ウ キャッシュメモリは，実記憶と仮想記憶のメモリ容量の差を埋めるために採用される。

エ 半導体メモリのアクセス速度の向上が著しいので，キャッシュメモリの必要性は減っている。

2） 処理装置で用いられるキャッシュメモリの使用目的として，適切なものはどれか。（基本 17 春 20）

ア 仮想記憶のアドレス変換を高速に行う。

イ 仮想記憶のページング処理を高速に行う。

ウ 主記憶へのアクセス速度とプロセッサの処理速度の差を埋める。

エ 使用頻度の高いプログラムを常駐させる。

≪解答≫

1） ア　イやウはキャッシュではなく、仮想記憶（主記憶と仮想記憶の間）の記述であり、仮想記憶の記述として正しい。

2） ウ　アやイは仮想記憶に関する記述である。アのアドレス変換の高速化は TLB。イは特には無い。