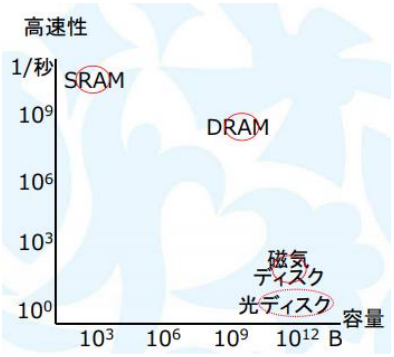
アーキテクチャ第１１回予習資料　　第１１回の授業までに空欄を埋めて予習。　質問は事前＋当日受付けます

第11回　メモリの高速化、キャッシュ、インターリーブ

11－1.　メモリ階層の概念



メモリは、高速にすると大容量にならない、逆に大容量にすると高速にならない、という関係があると言われます。 右図がいくつかのメモリ素子についての値です。
昔から、何とかしてこの問題を回避しようと、アイデアが練られてきました。今回はその技術を学びます。
（注意：　実は、高速性と大容量が両立しないというのは、100%いつも正しいわけではありません。ある技術が進んだ時、右図の関係が崩れるようなデバイスが出てくることがあります。しかし大抵の場合は、他の技術も同じように進んで、右図と同じような関係が再び成り立つようになってしまうようです。 もちろん、まったく新しい考え方が出てくれば、変わるのかもしれませんが。）

a)　用途別の使い分け（適材適所）

今まで見てきたコンピュータの仕組で、データを記憶する場所として、メインメモリと（汎用）レジスタが出てきています。
レジスタは　（　　）の中にあって、アクセス速度は（　　）ですが、容量は（　　）メモリです。
メインメモリは、CPU とは独立した回路で、アクセス速度はやや（　　）ですが、容量は（　　）メモリです。
これらをプログラムの工夫で使い分けることができます。
右の2つの例を比較してみましょう。上の例はメモリ上の変数 S に対して i を順番に加えています。下の例はその代わりに汎用レジスタ GR3 上に i を順番に加えて、最後にメモリ上の S に書き込んでいます。

```
メインメモリ上の S を 0 にする
for (i=0; i<100; i++){
   メモリ上の S を GR3 に読出す
   GR3 に i を加えて GR3 に戻す
   GR3 をメモリ上の S に書き戻す
}
```

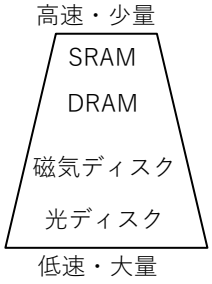
```
GR3 を 0 にする
for (i=0; i<100; i++){
   GR3 に i を加えて GR3 に戻す
}
GR3 をメモリ上の S に書き込む
```

上の例では、ループを1回実行するごとに（低速な）S への（　　と　　）をするのに比べて、下の例ではループを回っているときは（高速な）汎用レジスタ GR3 へ足しているだけです。その分実行は（　　い）です。
広い目で見ると、低速なメインメモリと高速なレジスタがあるとき、ループで何回もアクセスするところでは（　　）に変数を置いておくことによって、プログラムの高速化が図れる、ということになります。 しかし一方で、レジスタは①量が（　　）、実際にCOMET－Ⅱではの GR は（　　）個、また、②外部（入出力等）とやりとりできない、という欠点があるので、レジスタをうまく使い回して高速性を稼ぐという工夫が必要になります。

b）メモリ階層　（hierarchy）

また、メインメモリとファイル（二次記憶）との関係でも、適材適所の考えが成り立ちます。 上記の例と同じことを考えてみましょう。（実際にはこのようなやり方は性能上絶対にやりませんが。） 変数 S をハードディスク上のファイルに置きます。 上記と同じようにループで i を S に足してゆきます。1回ループするごとに、ハードディスクを読んで、i を足して、結果をハードディスクに書き込みます。ハードディスクはメインメモリ（DRAM）に比べて格段に遅いので、時間がかかるようになります。

つまり、レジスタ（ RAM） ⇔ メインメモリ（ RAM） でも メインメモリ（ RAM） ⇔ ファイル（ハードディスク＝磁気ディスク） でも、同じような関係が成り立つと言えます。この様子を、右図のように「階層的」に理解することができます。 これを「メモリ階層」と呼ぶことがあります。



11－2 キャッシュ

a) キャッシュの考え方

キャッシュは、「メインメモリを見かけ上（　　）する」仕組です。

考え方（駅前 KIOSK 本屋のストーリー）は、スライドを見て理解してください。

スライドでの KIOSK のストーリーをひとことにまとめると、

＊大型の本屋のように本を揃えたいが、駅前 KIOSK にはスペースがないので置けない。

＊駅前 KIOSK には（　　）を置き、離れた倉庫には（　　）を置く。

＊動きは、客の買いたい本が KIOSK にあればすぐにそこで渡せる。 なければ離れた倉庫に取りに行き時間がかかる。

＊客の待ち時間の平均値 Tav は、すぐに渡せるときの時間 Tc、倉庫へ取りに行く時間 Tm、すべての客のうちすぐに渡せる客の割合を h とすると、

＊もし、すぐに渡せる客の割合 h が非常に大きければ、Tav は（　　）に近づく。つまり、ほとんどの客の待ち時間は（　　）となって、あたかもすべての客が KIOSK でサービスされているような状況にできる。

＊すぐに渡せる客の割合 h を多くするには、（どのような　　）本を KIOSK に並べればよい。

これをそっくり、メインメモリのアクセスに置き換えます。

駅前 KIOSK 本屋　⇒ （　　）

少し離れた倉庫　⇒ （　　）

すべての客のうちすぐに渡せる客の割合 h　⇒ （　　）のうち（　　）の割合 h

すぐに渡せる割合を多くするには、

KIOSK には（　　）本を並べる　⇒ （　　には　　）データを並べる

b） キャッシュのアクセス時間のモデル

上記で計算したように、メインメモリのアクセス時間を Tm、キャッシュのアクセス時間を Tc とすると、

アクセスにかかる平均時間 Tav　＝　（　　）

で表されます。

これに、数字を当てはめてみましょう。 まずは右の場合にアクセス時間の平均値を計算してください。時間の単位は Tc を 1 とします。

	かかる時間	確率
キャッシュで済む場合	1	0.5
済まない場合	10	0.5

もう1つ、別の数字を考えてみましょう。この場合、アクセス時間の平均値はどうなるでしょうか。

	かかる時間	確率
キャッシュで済む場合	1	0.9
済まない場合	100	0.1

では、逆の計算をしてみます。かかる時間の比率が Tc：Tm＝1：1000 のとき、平均アクセス時間 Tav が Tc の 1.5 倍になるためには、キャッシュで済む確率（h）がいくつでなければならないでしょうか？

なお、スライドの追加部分にあるように、ランダムにメモリをアクセスすると、h はあまり大きくならず、Tav は Tc よりはるかに大きくなりますが、アクセスのパターンなどの影響で、実際には h はかなり 1 に近く、Tav は Tc に近い値になります。

11－3　キャッシュのマッピング

スライドで細かく説明しているので、それを見てください。 要約すると

＊　キャッシュは、メインメモリの一部をブロック単位（128 バイトとか）でコピーして持ってくる。

＊　メモリ内のブロックを、キャッシュの中のどこに置くか、が「マッピング」の問題。 2つの考え方ができる。

＊　考え方①　＝　（　　）方式　＝　（やり方　　）

　　場所の効率はよい、 ブロックを探すのが大変

　考え方②　＝　（　　）方式　＝　（やり方　　）

　　他が空いているのに割当てがぶつかってしまう（前にいたブロックを追い出さなければならない）

　　ブロックを探すのは簡単

＊　で、実際はこの2つの折衷案が使われる。（　　）方式と呼び、

　キャッシュの中を行に分けて、1つの行はたとえば4ブロックを置けるようにしておき、

　行の中で場所を選ぶのは（　　）方式で、どの行を選ぶかは（　　）方式で行う。

11－4　キャッシュの追い出し

スライドで細かく説明しているので、それを見てください。 要するに

＊　キャッシュのブロックは、追い出しが必要なことがある。

　CPU がメモリアクセス要求　⇒　キャッシュのあちこちにブロックのコピーがたまる

　　⇒　次に別のブロックをキャッシュへ持ってくるときに、入れる場所が無い

　　フルアソシアティブ方式ならすべて満杯になった時

　　ダイレクトマッピング方式なら変換した場所に既に使われている時

＊　誰を追い出すか

　ダイレクトマッピング方式の場合　⇒　（　　）

　フルアソシアティブ方式の場合　⇒　誰を追い出すか選ばなければならない　⇒　選び方？

　　①　ランダム　（　　）を選ぶ

　　②　到着順（FIFO）　（　　）を選ぶ

　　③　最長未使用時間順（LRU）　（　　）を選ぶ

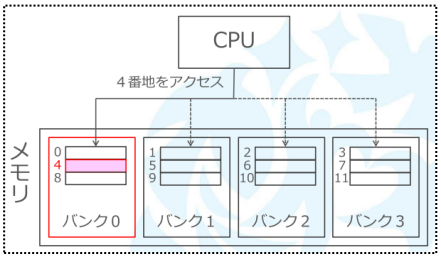
　　④　将来使わないものから優先（OPT）（将来のことが分からないので、実現不可能）

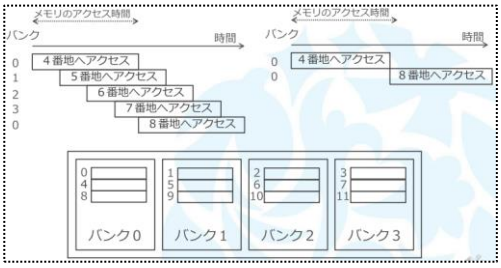
　後で使うブロックを追い出してしまうと、ヒット率が下がり、性能が低下する。

11－5　インターリーブ

メモリをバンクに分ける

異なるバンクは同時アクセスが可能





アドレスの最下位 n ビットでバンクへ振り分けるようにする。そうすると、

CPU から連続したアドレスを要求された時には、異なるバンクに行くので、並行してアクセスできる。

（例） バンク数＝4 とし、アドレスの下位 2 ビット（00, 01, 10, 11）でバンクを区別するように振り分ける。

　CPU から、アドレス 4,5,6,7 と連続してアクセスした場合、それぞれのアクセスはバンク 0,1,2,3 に行く。（並行できる）

　メモリの読出し時間の 1/4 の間隔で 4,5,6,7 へのアクセス要求を出せる。 見かけ上、メモリが 4 倍速くなる。

　アドレスが 4,8,12,16 のように同じバンクへ続くと、アクセスは直列にしかできないので、待たされて、遅くなる。