インターフェイス設計論

鳥取大学工学研究科
菅原 一孔

メモリと記憶装置

メモリと記憶装置

コンピュータシステムにおいて重要な構成要素：メモリ

基本メモリ方式：物理メモリ，仮想メモリ，キャッシュ

プログラマのメモリの想定⇒メインメモリに焦点
アーキテクトの観点
　　⇒データを保存するための半導体ディジタル装置

メモリの技術と構造

- メモリ技術の特徴
  - メモリの揮発性

    電源停止後のデータの保持
  - メモリアクセス

    ランダムアクセスと逐次アクセス（FIFO）
  - データの永続性

    データが取り出されるのか，更新されるのか

    ROM，ProgrammableROM，

    ElectricallyErasablePROM，FlashROM
  - １次，２次メモリ

    コンピュータの高速で揮発性の内部メインメモリと低速な外部不揮発性記憶装置

- 命令メモリとデータメモリ

  フォン・ノイマンアーキテクチャ

  ⇒プログラムとデータを1つのメモリに保持

  命令メモリ，データメモリの分離

  フェッチ等でのアクセス頻度高

  ⇒命令メモリはデータメモリより高い性能が要求される

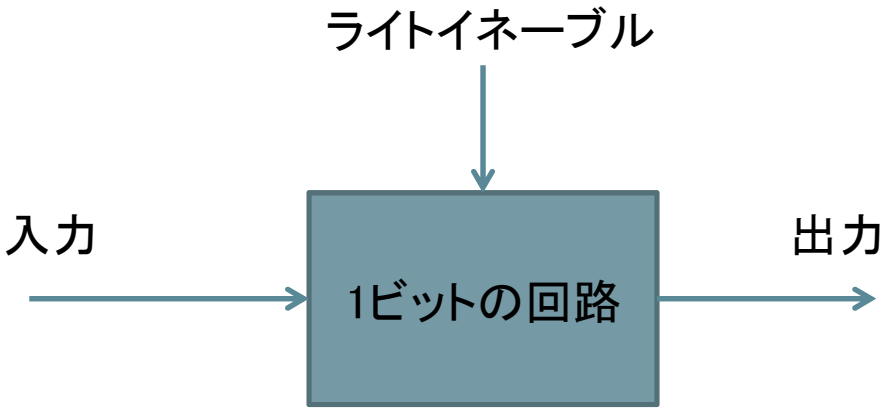
物理メモリと物理アドレス

- コンピュータメモリの特徴

  メインメモリ：RAM

    揮発性，読み込みと更新

- StaticRAMとDynamicRAM
  - SRAM:1ビットをフリップフロップなどの，複数のトランジスタで構成された小規模なディジタル回路で構成
  - 高速な動作
  - 電力消費，熱放出での課題
    - 連続して動作する多数のトランジスタが存在



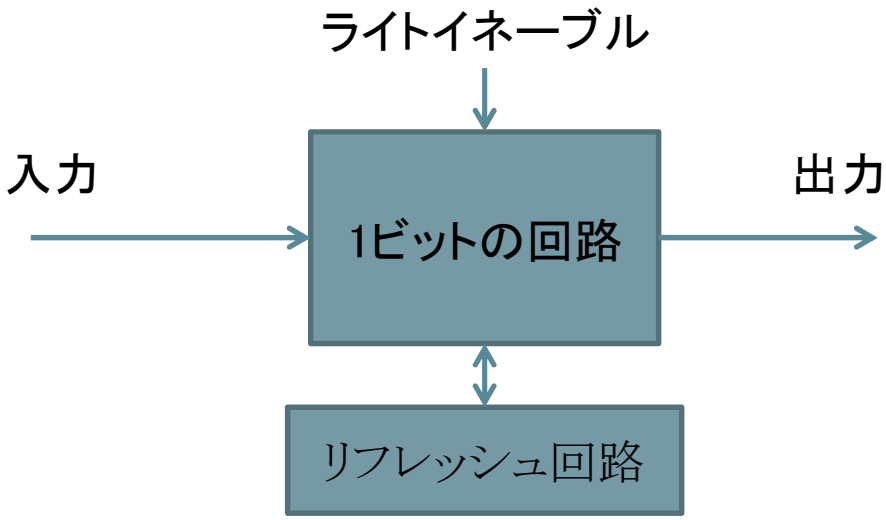
DRAM：電荷を保持するキャパシタの充放電により
　　データを書き込む

電荷の放電により情報の消滅

　　⇐ リフレッシュ機構が不可欠

リフレッシュ機構：全メモリを巡回する小さい回路規模

標準的なメモリ操作との併用が不可欠



- 容量

  半導体の単位面積当たりのメモリセルの数

  ⇒標準的な大きさのチップ上に格納できるビット数

- 読み込みおよび書き込み性能の分離

  リード操作性能：読み込みに要する時間

  ライト操作性能：書き込みに要する時間

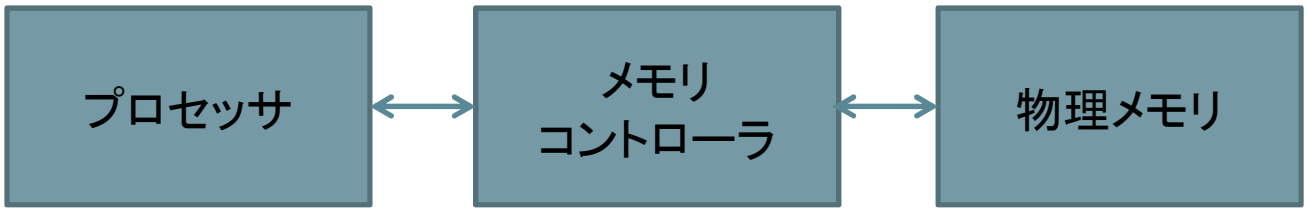
- 遅延とメモリコントローラ

  メモリサイクル時間

    プロセッサがメモリアクセス要求を出してから処理が完了するまでの時間

      読み込みサイクル時間　tRC

      書き込みサイクル時間　tWC



- 同期メモリ技術
  プロセッサのクロックとメモリのクロックの不一致
  ⇒遅延時間の増大
    同期メモリ
      システムクロックに同期したメモリ動作
        SynchronizedDRAM
        SynchronizedSRAM

- マルチデータレートメモリ

  多くのコンピュータシステム

  ⇒メモリの動作速度がボトルネック

  メモリシステムの高速化によりシステム全体の性能改善

  高速データレートメモリ技術

  DoubleDataRate:標準的なクロック速度の2倍で動作

  QuadrupleDataRate:　〃　4倍で動作

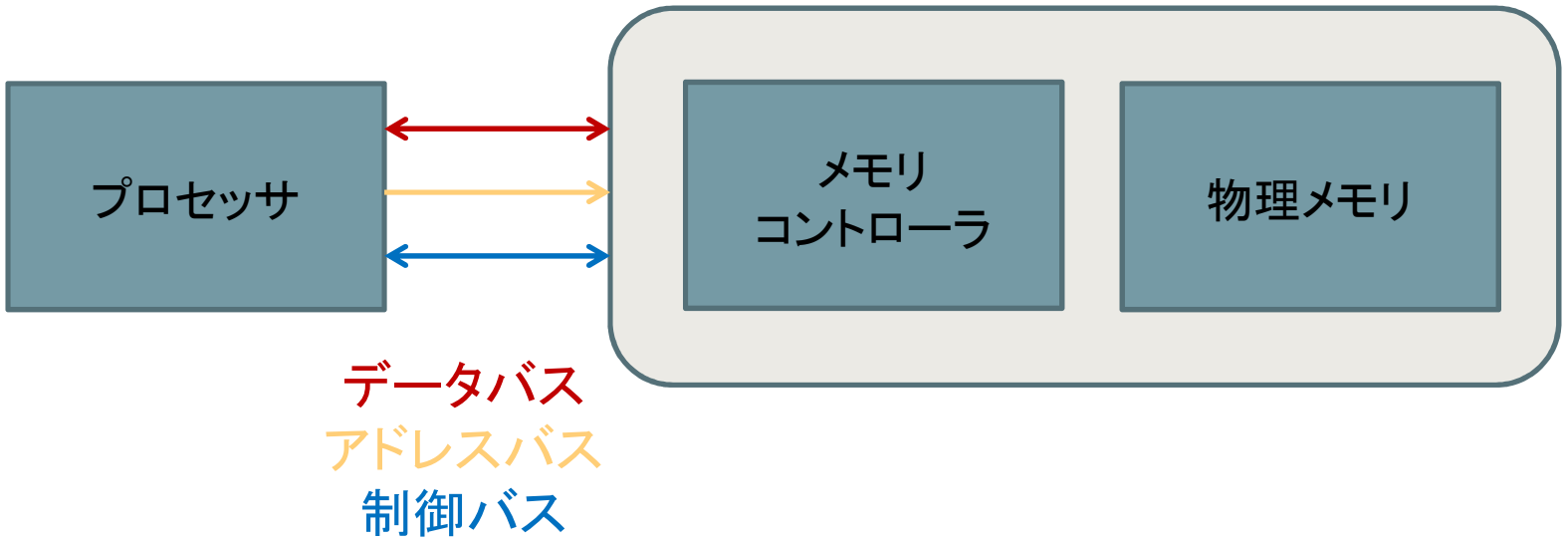
- メモリ技術の例

技術	意味
DDR DRAM	Double Data Rate Dynamic RAM
DDR SDRAM	Double Data Rate Synchronized Dynamic RAM
FCRAM	Fast Cycle RAM
FPM DRAM	Fast Rage Mode Dynamic RAM
QDR DRAM	Quad Data Rate Dynamic RAM
QDR SRAM	Quad Data Rate Static RAM
SDRAM	Synchronized Dynamic RAM
SSRAM	Synchronized Static RAM

- メモリ構成
  どの技術を利用するのか
  どのように構成するのか

  ハードウェアの内部の構造
  メモリがプロセッサに示す外部アクセス方法

- メモリアクセスとメモリバス



- メモリ転送サイズ

  1操作により，読み込み，書き込みが行われるデータ量

- 物理アドレスとワード

  物理メモリをNビットごとのブロックに分けて管理

  N:メモリ転送サイズ

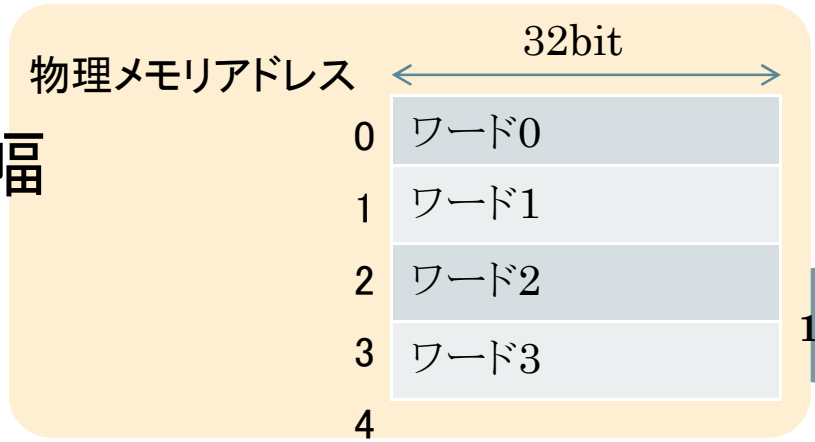
  ワード(語):Nビットのブロック

  転送サイズ:

  　ワードサイズ，ワード幅

  物理メモリアドレス

  ワードアドレッシング



- 物理メモリ操作

  物理メモリハードウエア：1ワード単位での読み書き

  ⇒メモリ転送サイズ

  リード/ライト操作はワード単位で適用

- ワードサイズと他のデータ型

  データを格納するためのメモリ

  ⇒通常のデータを収納

  ⇒整数を保持するのに十分な大きさ

  プログラムを格納するためのメモリ

  ⇒頻繁に利用される命令を格納

  並列動作するためのメモリ

  例）32ビットのメモリワードサイズ

  標準的な整数，単精度の浮動小数点

  ⇒32ビット

- 極端な場合:バイトアドレッシング

  バイト単位にアドレスが割り当てられている

  文字列のような小さなデータに容易にアクセス可能

  ⇔プログラミングの利便性が高い

  ワードアドレシングに比べ多くのアドレスが必要

- ワード転送によるバイトアドレッシング

  一般に，ワードアドレッシングはバイトアドレッシングに比べ一度の多数ビットの読み書きを行う

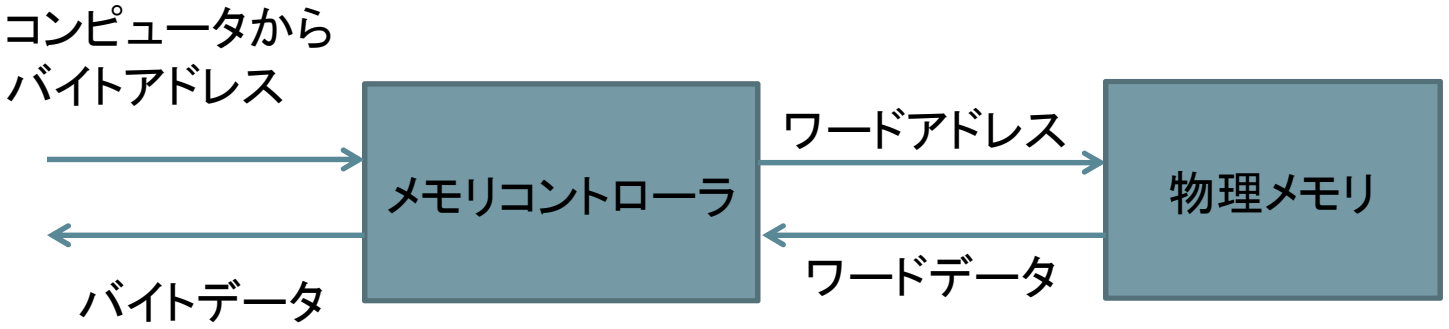
  ⇒高速処理が可能

  ワードアドレッシングの高速性とバイトアドレッシングのプログラミング利便性の両立

  ⇒2つのアドレッシングの変換

  ⇒高速なメモリコントローラの導入

メモリコントローラ【読出し操作】

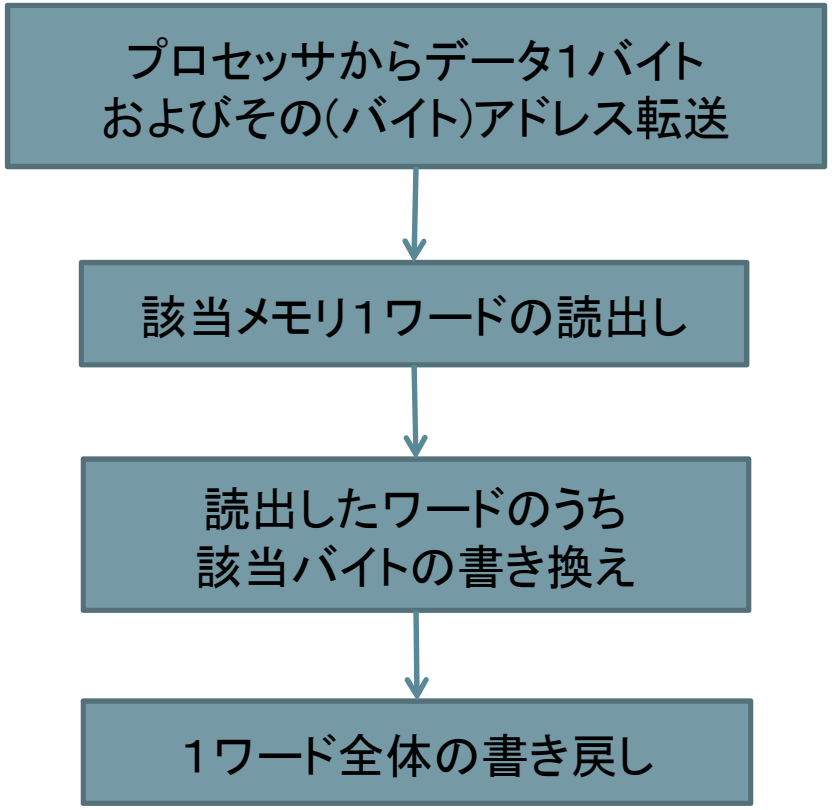


32bit

0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

ワードアドレス　　バイトアドレス

- メモリコントローラ【書込み操作】



- メモリコントローラ

  1ワード＝Nバイトのシステムにおけるアドレス変換

  バイトアドレス：B

  対応するワードアドレス：W

  ワード内のバイトオフセット:O

$$W = \left\lfloor \frac{B}{N} \right\rfloor$$

$$O = B \bmod N$$

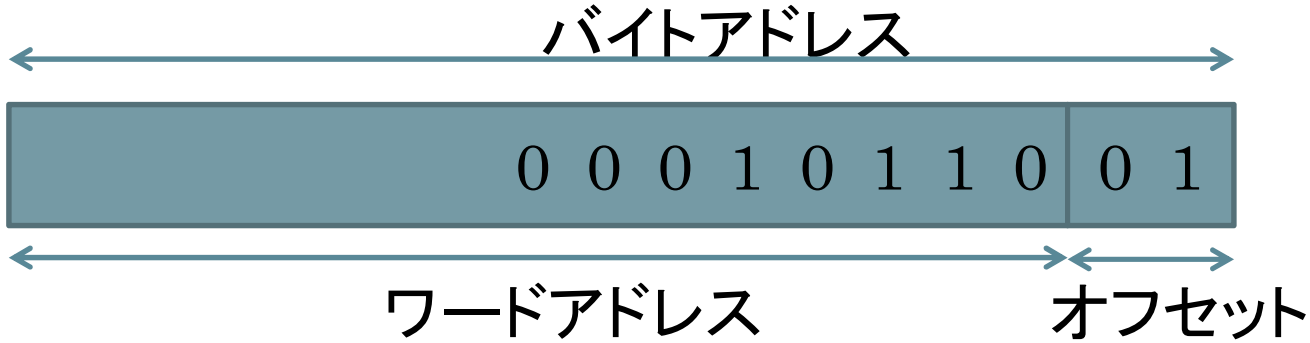
例）N=4の時，バイトアドレス11は，

ワードアドレス2，オフセット3に対応

- 2のべき乗を使う意味
  アドレス変換には割算が必要
  割算を実行し，余りを計算するには計算時間や，
  余分なハードウエアが必要

  Nに2のべき乗を使用すると算術演算を回避可能

  例）N=4の場合



- バイト整列とプログラミング

整数のバイト＝物理メモリのワードの場合
バイト並び：バイト 12，13，14，15で構成された整数
⇔整列している
バイト 6，7，8，9で構成された整数
⇔整列していない

バイト並びが必須なアーキテクチャを持つプロセッサ
非整列なアドレスに対して整数をアクセス
⇒エラー発生
任意のバイト並びが可能なプロセッサ
⇒整列したアクセスより転送速度の低下

- メモリ容量とアドレス空間

  プロセッサは整数と同じビット数のアドレス空間

  32ビット整数を取扱うシステム

  ⇔32ビットのアドレス空間

  0から4,294,967,295番地までのアドレスを表現

  例）N=4の時

  バイトアドレッシング：

  4,294,967,296バイトのアドレス空間

  ワードアドレッシング：

  17,179,869,184バイトのアドレス空間

- ワードアドレッシングによるプログラミング

プログラマの利便性

⇒バイトアドレッシングのプロセッサ

メモリ空間の縮小(同一のアドレスバス幅ならば)

ワードアドレッシングのプロセッサ

1バイトのデータの読出し

該当ワードの読出しと，その中から該当バイトの読出し

1バイトのデータの書き込み

該当ワードの読出しと，指定バイトの更新および書き換えワードの書き込み

シフト演算

- ポインタとデータ構造

C言語

char *cptr;

変数cptrは文字型の(バイト単位の)ポインタ

メモリアドレスと同じサイズのメモリを割り当て

任意のバイトアドレスを格納可能

int *iptr;

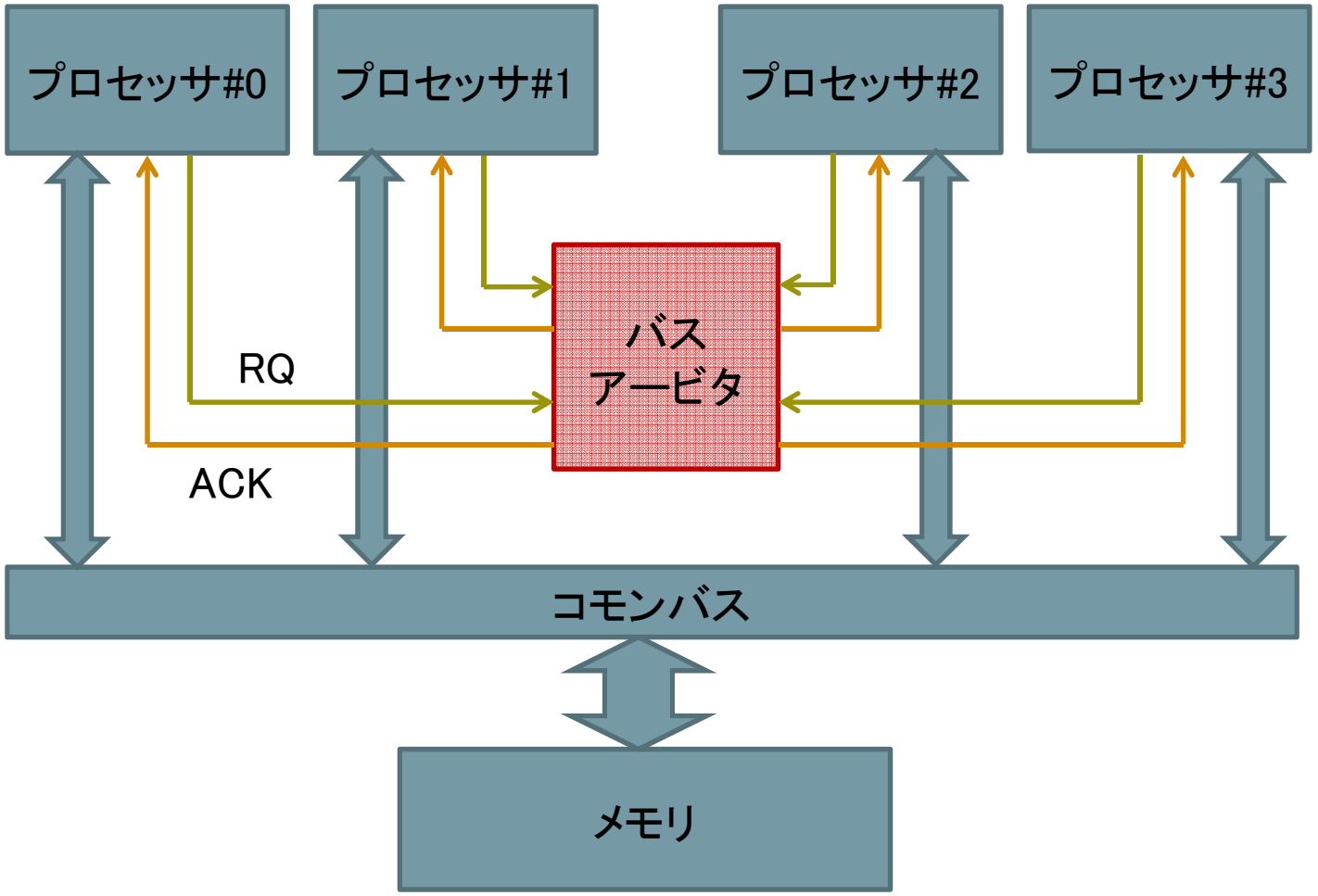
変数iptrは整数への(ワードへの)ポインタ

整数が4バイトの時,

iptr++により, iptrの値は4増加

インクリメント文により次のワードへの移動

- マルチプロセッサによるメモリアクセス



- プロセッサは，RQ信号線を使って調停機構(Arbiter:アービタという)に使用要求
- アービタは，複数のプロセッサから要求がある場合には，どのプロセッサにコモンバスの使用権を与えるかを決定し，ACK(Acknowledge)信号線を使って，使用権を与えるプロセッサには使用可(ACK)，その他のプロセッサには使用不可(NACK)という信号を返す
- 使用可の応答を得たプロセッサがコモンバスを使用してメモリにアクセス
- アービタは，到着している要求を見て，次のメモリ使用権をどのプロセッサに与えるかを決定

- アービタがどのように使用権を与えるプロセッサを決めるかは使用形態に依存
- 組み込みシステムのように各プロセッサの処理分担が決まっており，処理の優先度が固定されているというようなケース
- 汎用のプロセッサでは，内部にカウンタを持ち，使用権を与えるプロセッサをグルグルと順番に廻るラウンドロビンのように，全プロセッサに均等に使用権を与える方法を用いるのが普通

アクセス遅延(レイテンシ LATENCY)

- CPUがメモリをアクセスる際，まずアドレス信号を出力し，メモリ内のアドレスを確定したのち，指定されたメモリからデータを受け取る.
- 出力データがデータバスで確定するまでにかかる時間：アクセス時間
- データ転送において、データを要求してから実際に送られてくるまでの待ち時間

メモリバンクとインターリーブ

- メモリバンク

メモリコントローラがメモリを管理するときの単位となる，一定の容量を持ったメモリの集合．メモリの増設はバンク単位で行う必要がある．

コンピュータが内蔵するメモリはある程度の容量ごとにまとめて管理され，アクセス要求が発生すると対象となる領域を選択し，その領域に対してだけアクセスを行うことで効率を上げる．この管理単位をメモリバンク，あるいは単にバンクと呼ぶ．

メモリバンクとインターリーブ

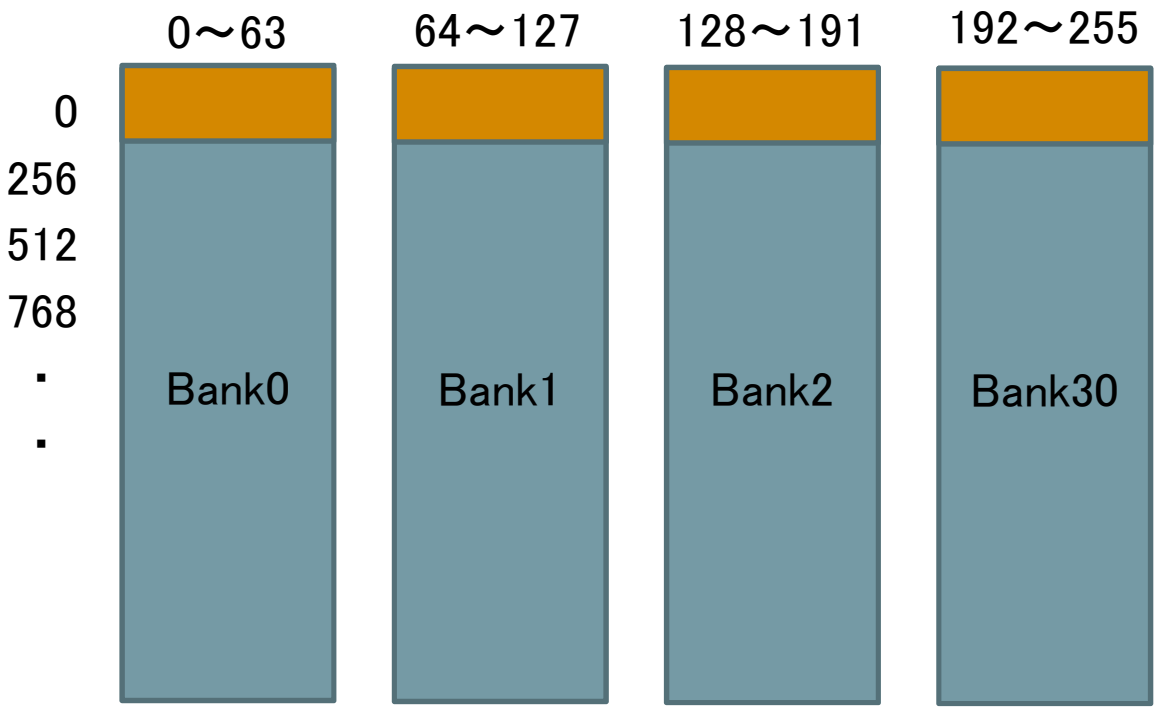
- インターリーブ

CPUがアクセス要求を行ってから実際にデータが送られてくる(あるいは書き込みが完了する)までにはレイテンシ(遅延)と呼ばれる時間差

メモリへのアクセスは時間がかかるため，コンピュータの処理速度はこの「待ち時間」に足を引っ張られている．レイテンシを短縮する試みは常に行われているが，CPU内の記憶素子との差は埋めがたく，また，低レイテンシのメモリは高価である．

一方，メモリへのアクセス要求は短期的には局所性が極めて強く，連続した領域に順番に読み書きを行うことが多い．この特徴を利用して，複数のメモリバンクにまたがって連続したアドレスを交互に振っておき，あるデータにアクセスする遅延時間の最中に次のアドレスへアクセス要求を発行して時間を有効利用するのがメモリインターリーブ

- 4バンクのメモリ構成 ⇔ 4ウエイインターリーブ



Bank0: 0～63 + 256*n
Bank1: 64～127 + 256*n
Bank2: 128～191 + 256*n
Bank3: 192～255 + 256*n

- 各プロセッサからアクセスするメモリ番地がランダムな場合

  各バンクへのアクセスの確率は均等とすると

  4バンクのメモリに対して2つの異なる
  　　バンクをアクセスする確率＝3/4
  3つの異なるバンクにアクセスする確率
  　　3/4*2/4＝3/8
  4つが全部違うバンクにアクセスする確率
  　　3/8*1/4=3/32

  4バンクに分割して，それぞれが独立にメモリアクセスを処理できるようにすると，平均的に2つから3つのメモリアクセスを並行して処理可能

- 連想メモリ

  メモリ技術とメモリ構成の融合
- 一般的なメモリ：ユーザーのアドレス指定

  ⇒アドレスに格納されたデータ
- 連想メモリ：ユーザーがあるデータワードを指定

  ⇒全内容からそのデータワードを検索

  データワードが見つかれば，そのワードが見つかった場所のアドレスを返す
- 連想メモリ：メモリ全体をひとつの操作で検索

  事実上，全ての検索用途においてRAMよりもずっと高速

  ⇔連想メモリはコスト大

  完全並行動作する連想メモリではメモリ内の全ビット毎に入力データとの比較回路が必要

  データワード全体の一致を探索 ⇒比較結果をまとめる回路も必要

  連想メモリの回路サイズは増大し，製造コストも増大

  それら比較回路はデータが入力される度に全体が動作するため，消費電力も増大

- ３値CAM
  - ２値連想メモリ（Binary CAM）は最も単純なタイプ
  - 検索ワードの内容を 0，1 の二種類の値で構成
  - ３値連想メモリ（Ternary CAM） は三番目の値として ″X″ あるいは ″気にしない（Don’t Care）″を格納されたデータワードに使用
    - 例えば，３値連想メモリに “10XX0” というワードが格納されていた場合，“10000”，“10010”，“10100”，“10110” のいずれとも一致
  - ２値連想メモリに比較して検索の柔軟性が向上
  - 三種類目の値を持つという回路構成上のコスト増大
  - 一般にこの三番目の状態はメモリ全体にマスクビットを用意することで実装

- 適用例

  連想メモリはコンピュータネットワーク機器でよく使われる. 例えば, スイッチングハブはパケットを一つのポートで受信すると, 内部テーブルにそのパケットのMACアドレスとポート番号を格納する.

  その後, データを送信する場合には, 送信先MACアドレスをそのテーブル上で検索してパケットを送り出すべきポート番号を得て, そのポートにパケットを送る. このMACアドレスのテーブルは一般に二値連想メモリで実装され, 高速検索によってスイッチングによる遅延を小さくしている.

  CPUのキャッシュ制御部や, メモリ管理ユニット内のある種のキャッシュ(TLB)にも連想メモリを使用している.

仮想メモリの技術と仮想アドレス

- 定義

  仮想メモリ(VM : Virtual Memory):

  物理メモリと物理アドレスの限界を克服する，アドレス空間とメモリアクセス手法を提供する手法

- 仮想的な例：バイト単位の番地付け

- 仮想メモリ
  - メモリ管理ユニット（MMU）
    - 高機能なメモリコントローラ
    - プロセッサに対して仮想アドレス空間を提供

プロセッサ：
仮想アドレスを生成

仮想メモリシステム：
独立した機構として
分類

物理メモリと仮想メモリの区別
- 物理アドレス：実アドレス
- 物理メモリにあるアドレスの集合：実アドレス空間

- 物理メモリシステムを多重化するインタフェース

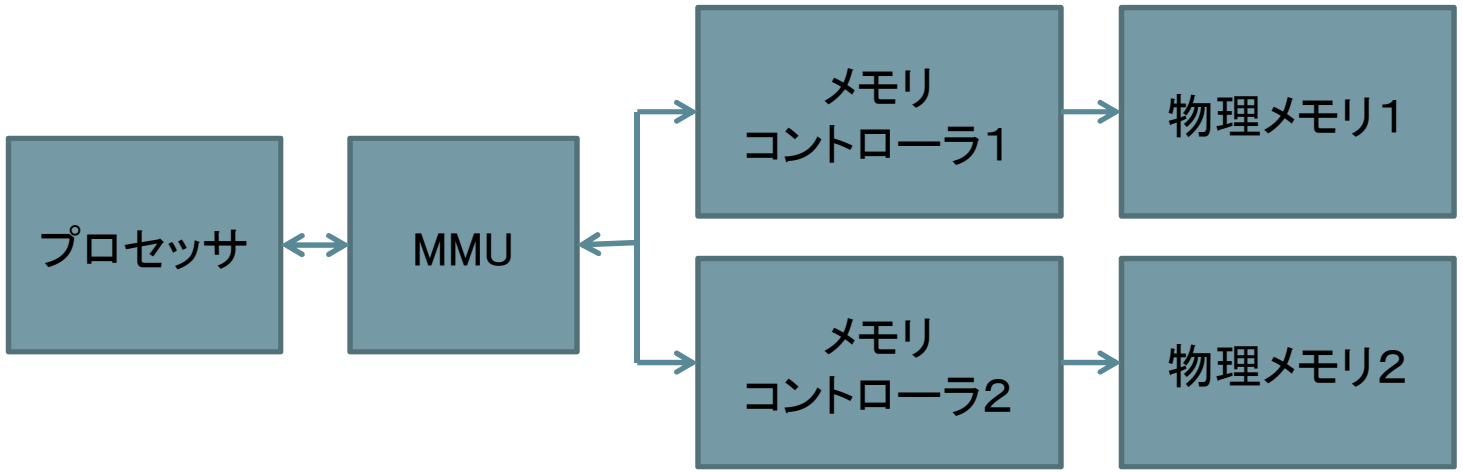
  複雑なメモリ構成

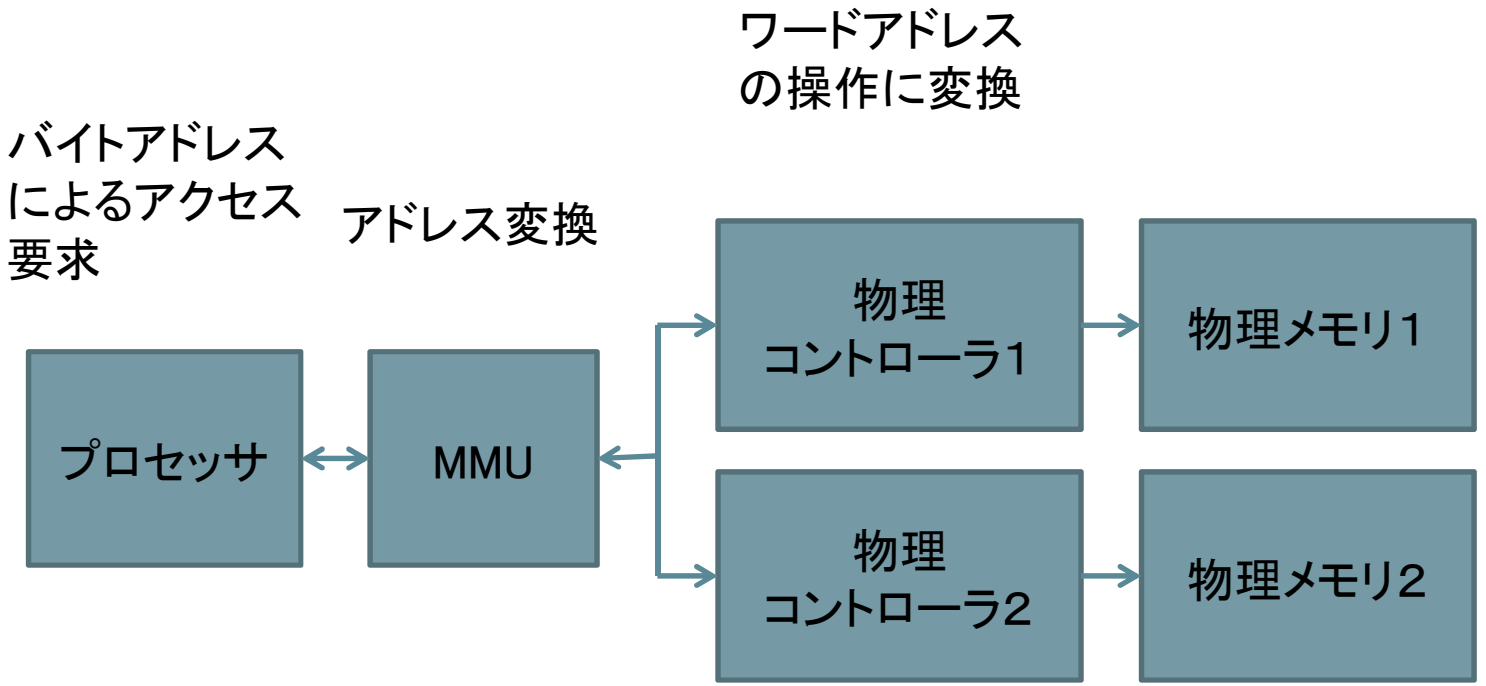
  ⇒単一の仮想アドレス空間として管理

  SRAMとDRAMの混在

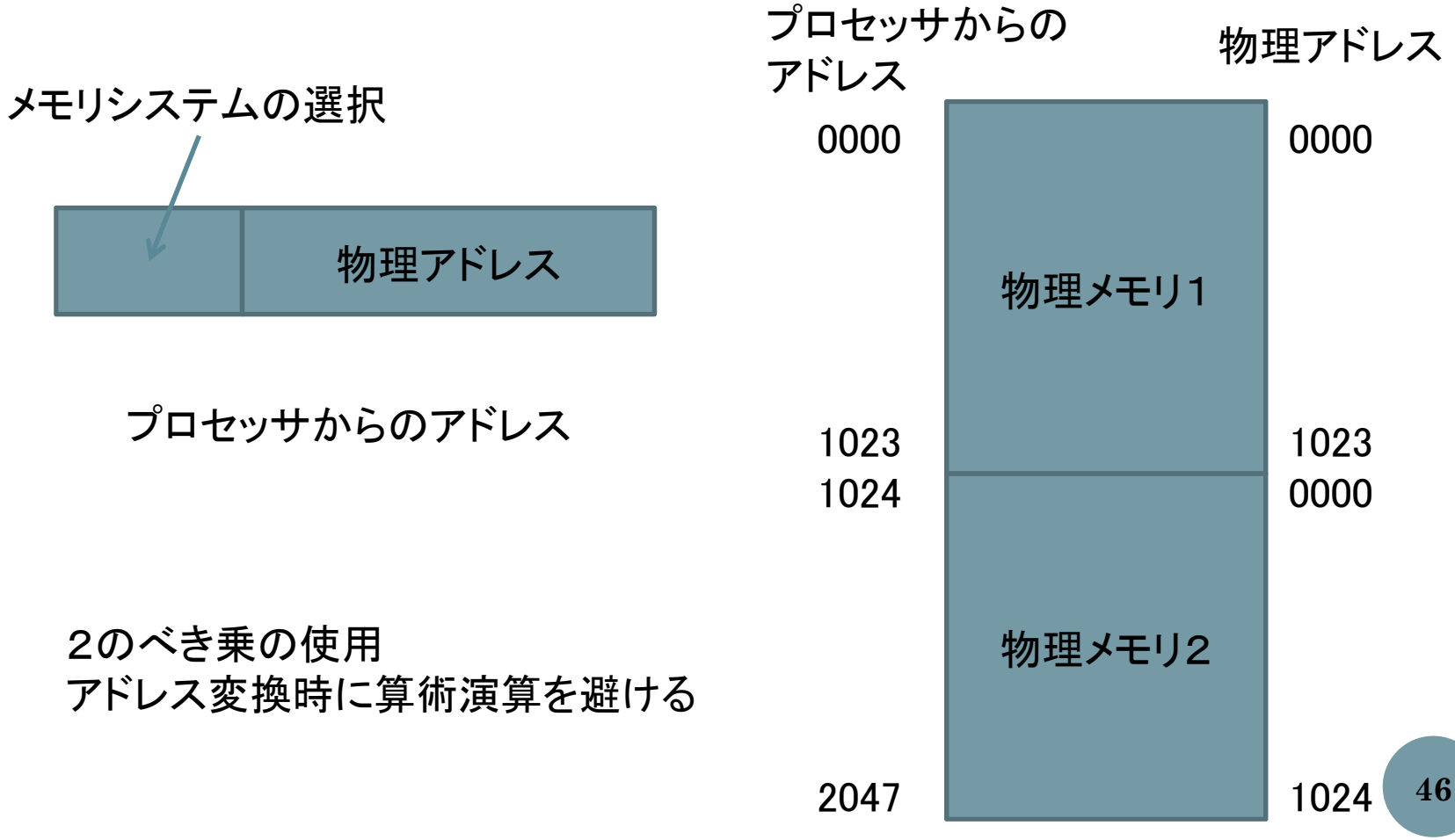
  1ワードに対し，異なったバイト数

  1word=4byte，1word=8byte

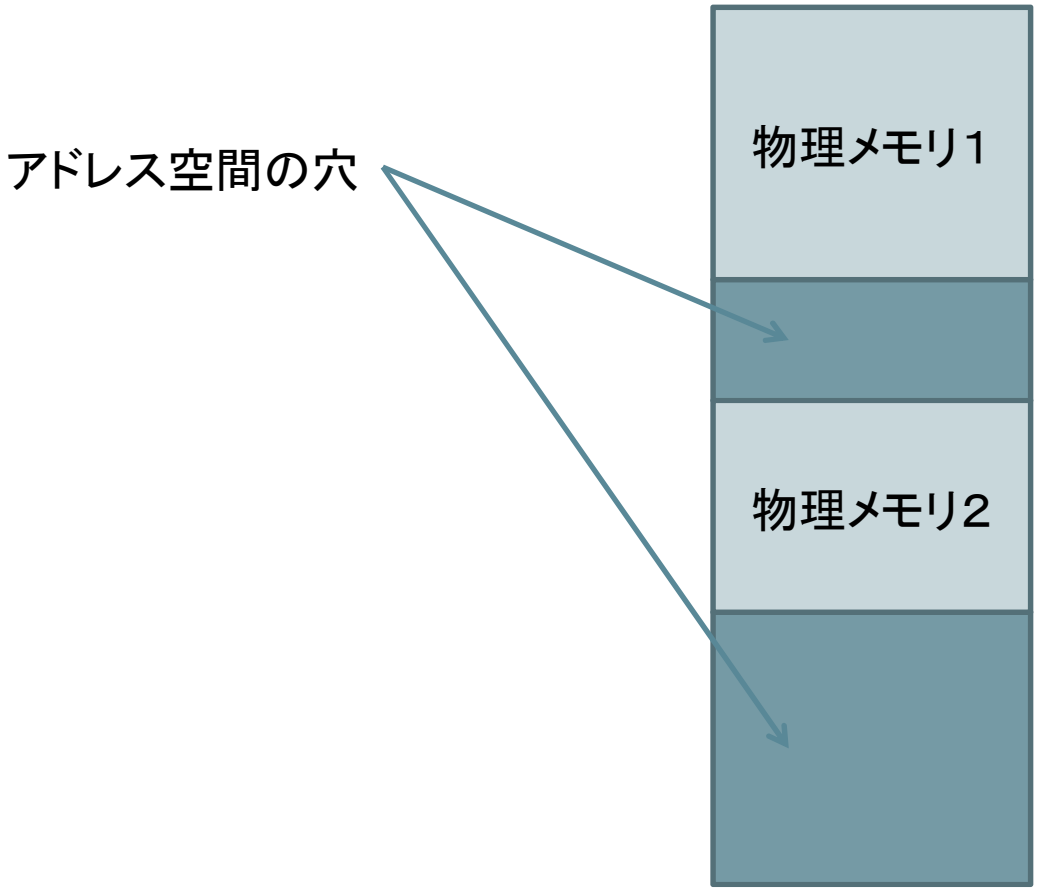



ワードアドレス
の操作に変換
バイトアドレス
によるアクセス
要求
アドレス変換
プロセッサ
MMU
物理
コントローラ1
物理メモリ1
物理
コントローラ2
物理メモリ2

アドレス変換（アドレスマッピング）



- 連続でないアドレス空間



- 他のメモリ構成

  仮想アドレスを物理メモリにマップする方法は多数の方式

  例）アドレスの下位2ビット：

  ４つのメモリモジュールにメモリをインターリーブ

  残りのビットはモジュール中のバイトの識別

  ⇒それぞれの物理モジュールを同時にアクセス可能なハードウエアの実現が可能

  ⇒アドレスが連続するバイトは別々のモジュール

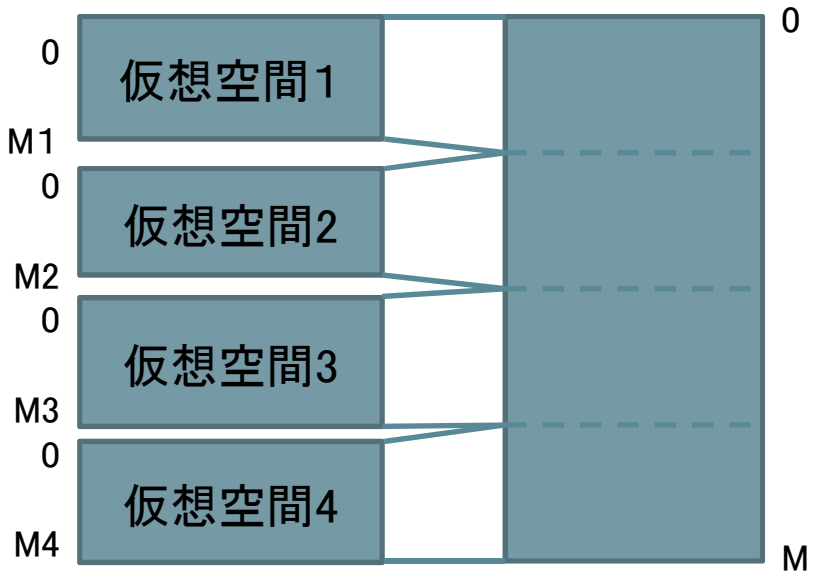
- より複雑な仮想メモリの仕組み
  - ハードウエアの均質な統合
    - 仮想の物理メモリの不均質を許す
    - 16ビットサイズのワード，32ビットサイズのワード
    - サイクルタイムの長短
    - RAMとROM
  - プログラミング上の便宜
    - 均質なアドレス空間への統一
    - ⇒個々のメモリに対する特別な命令の使用を避ける
    - ⇒メモリプログラムの書き換えが不要
  - 多重プログラミングの支援
  - プログラムやデータの保護

- 複数の仮想空間と多重プログラミング

  同一アドレスへのアクセス⇒衝突の発生

  個々のプログラミングに対し独立した仮想アドレス空間の提供により回避しようとする試み

  ⇒1つのプログラムで利用可能なメモリが減少



- 仮想アドレス空間の動的な生成

  小規模，特定用途のシステム

  ⇒ハードウエアによるメモリマッピング

  汎用のコンピュータシステム

  ⇒実行時に動的にメモリマッピングを変更

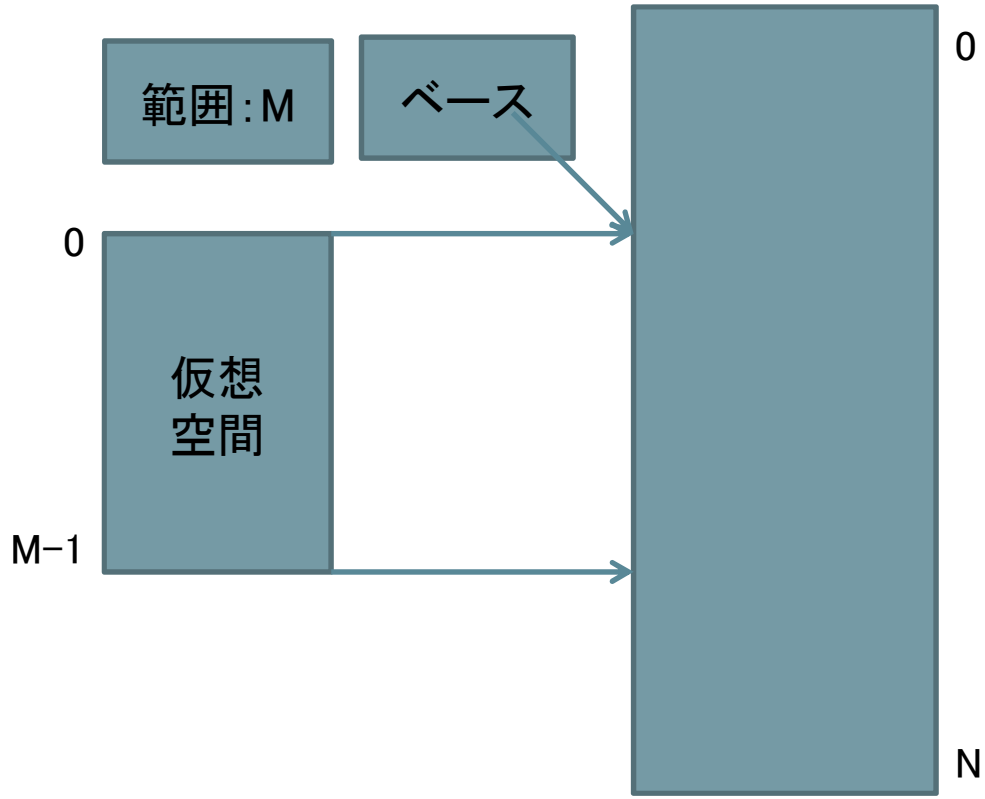
1. プロセッサ：リアルモードで立ち上がり

   （MMU利用なし，物理メモリを直接参照）
1. MMUへのマッピングの指定
2. 新しいマッピング上での実行モードの変更
3. MMUの活性化，特定番地への分岐
4. アプリケーションの実行

   （仮想アドレス空間へのアクセス）

- 仮想メモリシステムを構成する技術
  - ベース-範囲レジスタ
  - セグメンテーション
  - デマンドページング

- ベース−範囲レジスタ
  - 単一の仮想アドレス空間の生成
  - 物理メモリの領域にマッピング



- 仮想空間の変更（ベース–範囲レジスタ方式）

  ベース–範囲の仕組みは動的

  複数の仮想空間の間を移動可能

  例）OSが2つのアプリケーションプログラムA,Bを
  プログラムメモリにロード

  1. OS：リアルモードで稼働
  2. Aの実行準備が整ったとき，OSはAのメモリに対応するよう仮想メモリマッピング後，MMUを活性化
  3. Aのアプリケーションプログラムに分岐
  4. OSに制御が戻り
  5. Bについても同様

- 仮想メモリとベース–範囲と保護

  ベースレジスタ：仮想アドレスから物理アドレスへの
  マッピングの基本位置の設定

  範囲：プログラムが確保したメモリ空間を超えないよう
  保護

- セグメンテーション

  粗粒度のマッピング：すべてのアドレス空間をマッピングする仮想メモリ技術

  細粒度のマッピング：アドレス空間の一部をマッピング

  1. プログラムを可変サイズのブロックに分割
  2. プログラムが必要とするブロックのみをメモリにロード
  3. 他のブロックはディスク
  4. OSは未使用のメモリ領域に必要なブロックをロード
  5. ブロックの実行後，OSはブロックをディスクに移動
  6. メモリを解放

     ⇒メモリの断片化（フラグメンテーション）が発生

- デマンドページング
  - セグメンテーションを一般化した方式
  - セグメンテーションとの違い
    - ⇒どのようにプログラムを分割するか
  - プログラム＝一連の手続きの集合
  - セグメンテーション：
    - それぞれの手続きを保持するのに十分大きな可変サイズのセグメント
  - デマンドページング：
    - ページと呼ばれる固定サイズのブロック
    - （Pentium：1ページ4Kバイト）

- デマンドページングのハードウエアとソフトウエア

  デマンドページングを可能にする2つの技術

  - アドレスのマッピングを行い失ったページを検出するハードウエア
  - 外部記憶と物理メモリ間でページを移動するソフトウエア

  ハードウエアアーキテクチャがページングシステムを提供

  ソフトウエアが要求の処理を可能にする

- OSがMMUに設定
  - 仮想アドレスのどのページがメモリに存在するか
  - それぞれのページがどこに置かれているか
- 仮想アドレス空間を利用するプログラムを実行
- MMUは各プログラムからのメモリアクセスを変換
- 存在しないページへのアクセス：ページ違反

  ⇒OSへのページ違反の発生を通知
- ページへのアクセス要求発生時に動作
- ページ違反発生時：
  - 2次記憶上のページ，メモリ区画を特定
  - ページをメモリに読み込みMMUを再設定
  - ページをロード，アプリケーションプログラムの実行再開

- ページ置き換え

  複数のアプリケーションを実行⇒すべてのメモリを使用

  ⇒アプリケーションプログラムがページを参照

  ⇒OSはどのページがいつ参照されたかを把握

  長期に利用されていないページを外部記憶に退避

- ページングの用語とデータ構造

  ページ：プログラムのアドレス空間のブロック

  フレーム：ページを保持する物理メモリの区画

  cf) ソフトウエアがページを，メモリのフレームにロードする

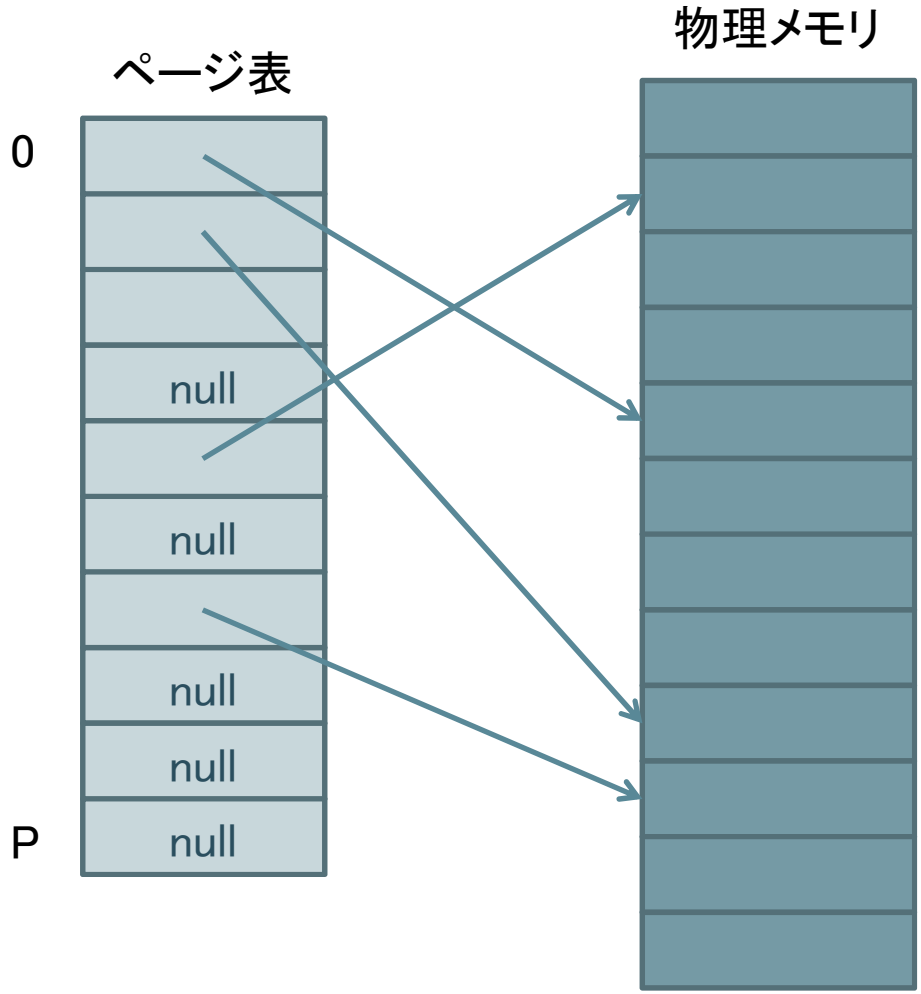
  ページがメモリ上にロードされているとき，ページは存在

  存在集合：

  現在メモリ上に存在するアドレス空間のすべてのページの集合

- ページ表

  デマンドページングに使われる1次的なデータ構造



ページングシステムにおけるアドレス変換

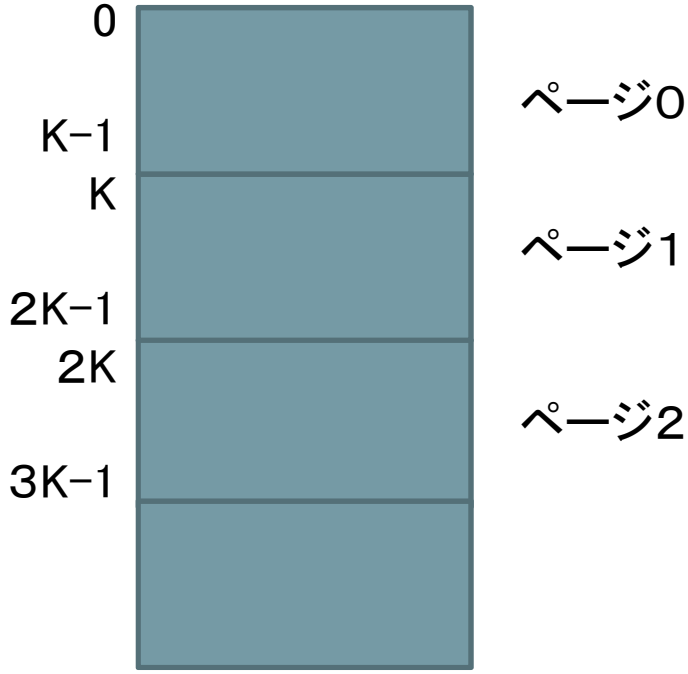
1ページ当たりのバイト数：K
仮想アドレス：V

ページ番号　$N = \left\lfloor \frac{V}{K} \right\rfloor$

ページ内オフセット

$O = V \bmod K$

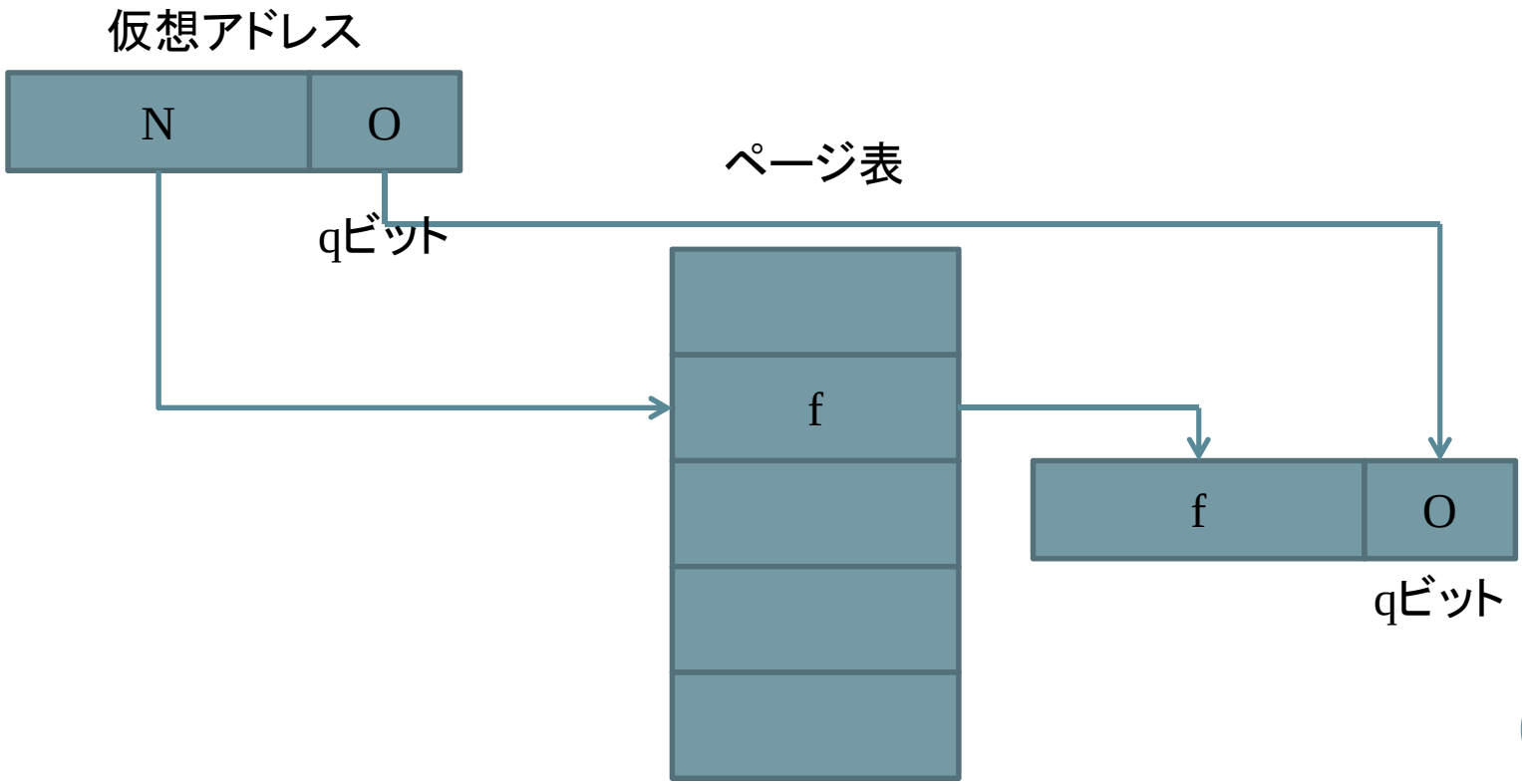
$V = pagetable[N] + O$



２のべき乗を使う

１ページ当たりのバイト数K：２のべき乗

$K = 2^q$:



- 存在ビットと使用ビット，それに変更ビット
  - ページ表上の制御ビット
    - 存在ビット
      - 現在メモリ上に当該ページが存在するかどうか
      - ソフトウエアにより設定，ハードウエアにより検査
    - 使用ビット
      - 一定期間にページが参照されたかどうか
      - MMUがページ表へのアクセス：使用ビットをセット
      - OS：定期的に検査
        - セットされていないとページ置き換えの対象
      - ビットのリセット

変更ビット

対応するページに書込み操作があったかどうか

ページングソフトウエア

ページのロード時に変更ビットをリセット

MMUは対応するページに書込み操作が行われた際変更ビットをセット

ページ置き換えの際，OSは

変更ビットがセットされているときは，外部記憶装置にページを書き戻し

リセットされているときは，書き戻し不要

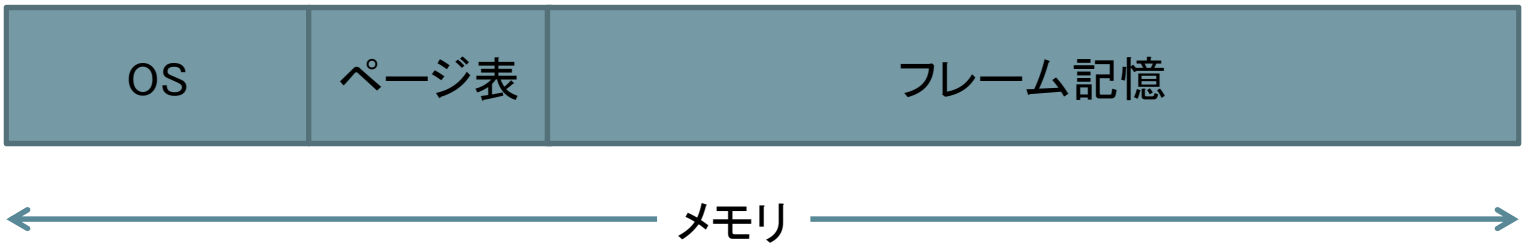
- ページ表の記憶
  1. プロセッサの外部のMMUチップ上に格納方式
  2. メインメモリ上に保持する方式

メモリ参照は処理の実行時に重要な役割

MMUの効率的な動作が求められる

SRAMにページ表

DRAMにフレーム記憶



- ページングの効果と変換側付きバッファ

  MMUが仮想アドレスを物理アドレスに変換するのに要する時間が重要

  OSがページ表を設定するのに要する時間より

- 変換側付きバッファ（TBL:Translation Lookaside Buffer）

  連想メモリの一種
  ページ表のエントリの服背を初期設定
  その後は，通常のアドレス変換と，TBLの高速な検索を並行実施
  プログラム実行時は連続した（あるいは近辺の）メモリフェッチが多数 ⇒ TBLにより連続した表引きを高速化

- プログラマに対する結果
  - プログラマが生成されるコードは，それぞれがページにフィットするよう編成
  - 文字列のようなオブジェクトは，連続したメモリ番地を占有
  - コンパイラはデータ項目をページに格納
- 配列のアクセス（2次元配列）
  - 列要素連続順で配列を確保
  - バイトの2次元配列の場合A[i,j]要素は，Qを列あたりの要素数として

    location(A)+i×Q＋j

⇒行要素連続順

- 列要素連続順の場合

```
for i=1 to N {
  for j=1 to M {
    A[i,j]=0;
}}
```

は

```
for j=1 to M {
 for i=1 to N {
   A[i,j]=0;
}}
```

に比べ高速な動作