東邦大学

生命の科学で未来をつなぐ

基本的な記憶管理
と
その手法２

前回のまとめ

- メモリ上にプロセスを並べて置く
- その結果、スペースの割当て管理が必要
- プロセスの大きさに合わせて（可変長）割当てると
  生成・消滅（＝確保・解放）を繰り返すうち
  切れ切れの無駄な空き地が溜まる
  （外部）フラグメンテーションと呼ぶ
- （外部）フラグメンテーションの起きる要因は
  ①可変長割当て（長さいろいろ）であることと
  ②くっつけられない（移動できない）こと

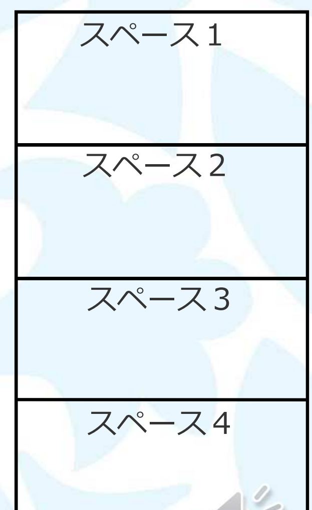
外部フラグメンテーションの要因を取り除くことを考える

- （要因１）可変長割当てであること
  - 可変長でなく、固定長割当てはできるか
- （要因２）移動して集められないこと
  - 何とかプロセスを移動する工夫は出来ないか？

東邦大学

固定長割当ては出来るか？

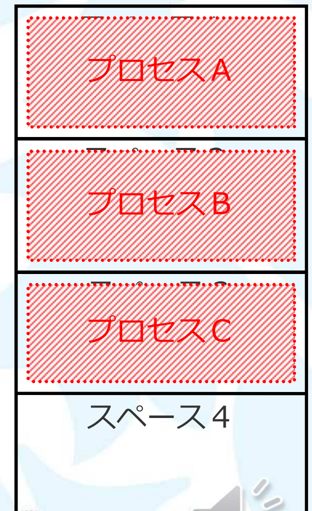
- 固定長割当ては、十分に可能
  - 要するに、最初にＮ個と決めておいて、メモリ全体をＮ等分すればよい



東邦大学

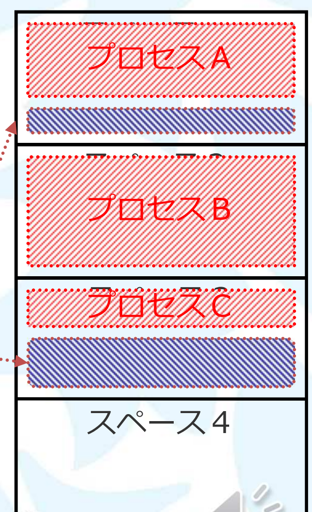
固定長割当ては出来るか？

- 固定長割当ては、十分に可能
  - 要するに、最初にＮ個と決めておいて、メモリ全体をＮ等分すればよい
- 管理は簡単 (長さを気にしなくていい)



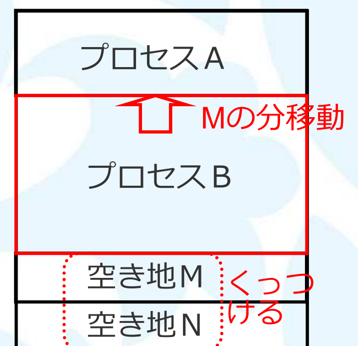
固定長割当ては出来るか？

- 固定長割当ては、十分に可能
  - 要するに、最初にＮ個と決めておいて、メモリ全体をＮ等分すればよい
- 管理は簡単 (長さを気にしなくていい)
- だが、プロセスが小さいと、区画内に余りのスペースができて無駄になる（内部フラグメンテーション）



区画（プロセス）は移動できるか？

- 移動して詰めることは可能
  - コンパクション（圧縮）とか
    ガベージコレクション(ごみ集め)
    とか呼ぶ

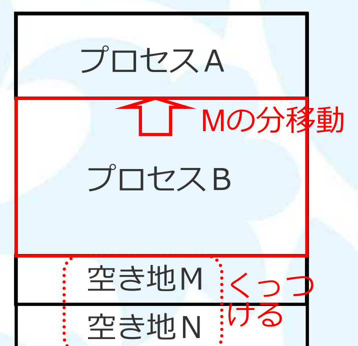


東邦大学

区画（プロセス）は移動できるか？

- 移動して詰めることは可能
  - コンパクション（圧縮）とか
    ガベージコレクション(ごみ集め)
    とか呼ぶ

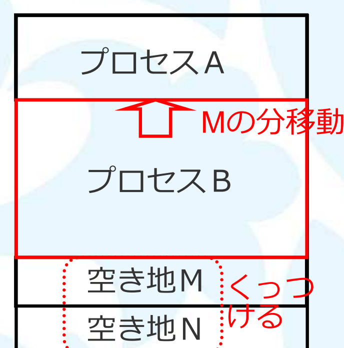
- プロセスの移動作業はやっかい



東邦大学

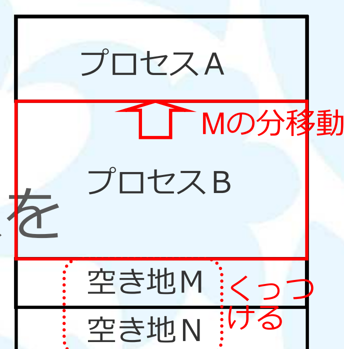
区画（プロセス）は移動できるか？

- 移動して詰めることは可能
  - コンパクション（圧縮）とか ガベージコレクション(ごみ集め) とか呼ぶ
- プロセスの移動作業はやっかい
  - メモリ内容をコピーする（手間はかかるが、やればできる）



区画（プロセス）は移動できるか？

- 移動して詰めることは可能
  - コンパクション（圧縮）とか ガベージコレクション(ごみ集め) とか呼ぶ
- プロセスの移動作業はやっかい
  - メモリ内容をコピーする
  - プログラム(命令)中の参照アドレスを移動に合わせて修正する必要



区画（プロセス）は移動できるか？

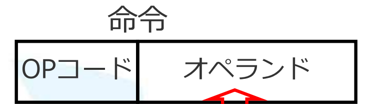
- 移動して詰めることは可能
  - コンパクション（圧縮）とか
    ガベージコレクション(ごみ集め)
    とか呼ぶ
- プロセスの移動作業はやっかい
  - メモリ内容をコピーする
  - プログラム(命令)中の参照アドレスを
    移動に合わせて修正する必要

非常にやっかいで、
実行開始後の移動はほとんど非現実的

（脱線）命令中のアドレスの書換えはやっかい

- 命令中のオペランド（覚えていますか）

（脱線）命令中のアドレスの書換えはやっかい



この内容はアドレス

- 命令中のオペランド （覚えていますか）
  - 操作される対象を指定
    例えば、演算命令の対象の変数
    　正確にはメモリ中の変数のアドレス

（脱線）命令中のアドレスの書換えはやっかい

この内容はアドレス

- 命令中のオペランド （覚えていますか）
  - 操作される対象を指定
    例えば、演算命令の対象の変数
    　正確にはメモリ中の変数のアドレス
    例えば、ジャンプ命令の飛び先アドレス

（脱線）命令中のアドレスの書換えはやっかい

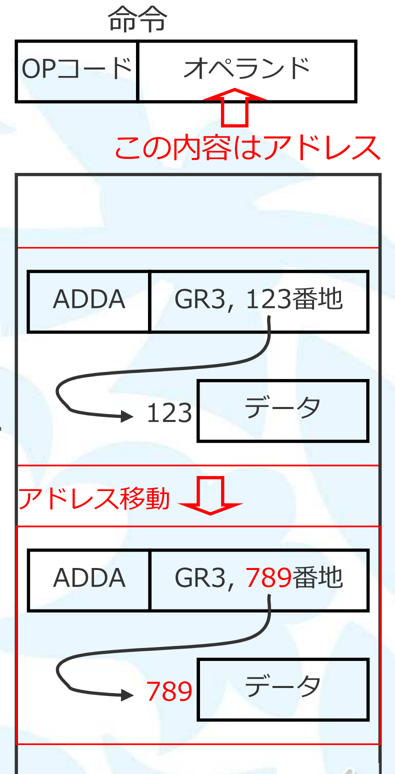
- 命令中のオペランド （覚えていますか）
  - 操作される対象を指定
    例えば、演算命令の対象の変数
    　正確にはメモリ中の変数のアドレス
    例えば、ジャンプ命令の飛び先アドレス
- プログラムのメモリ内の位置を動かす

（脱線）命令中のアドレスの書換えはやっかい

- 命令中のオペランド （覚えていますか）
  - 操作される対象を指定
    例えば、演算命令の対象の変数
    　正確にはメモリ中の変数のアドレス
    例えば、ジャンプ命令の飛び先アドレス
- プログラムのメモリ内の位置を動かす
  - 変数の置いてあるアドレスが変わる

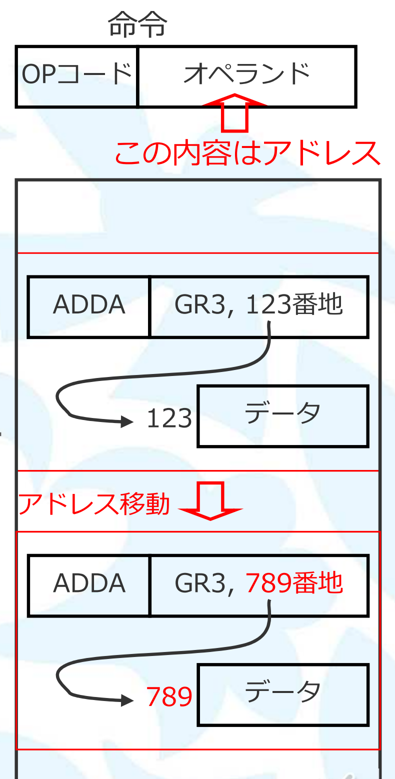
（脱線）命令中のアドレスの書換えはやっかい

- 命令中のオペランド （覚えていますか）
  - 操作される対象を指定
    例えば、演算命令の対象の変数
    　正確にはメモリ中の変数のアドレス
    例えば、ジャンプ命令の飛び先アドレス
- プログラムのメモリ内の位置を動かす
  - 変数の置いてあるアドレスが変わる
    ⇒　演算のオペランドを修正する必要



（脱線）命令中のアドレスの書換えはやっかい

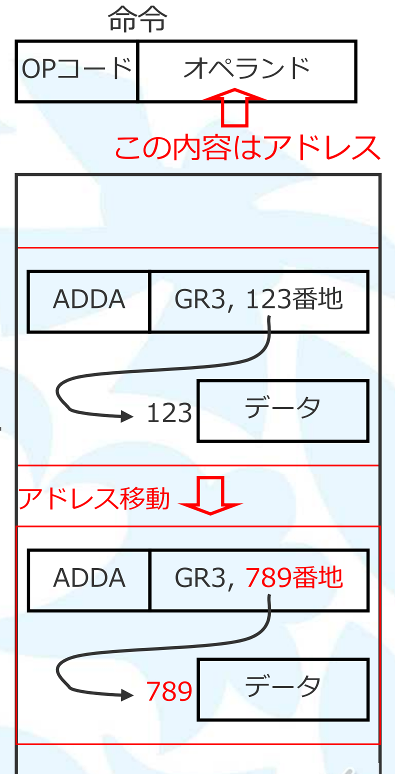
- 命令中のオペランド （覚えていますか）
  - 操作される対象を指定
    例えば、演算命令の対象の変数
    　正確にはメモリ中の変数のアドレス
    例えば、ジャンプ命令の飛び先アドレス
- プログラムのメモリ内の位置を動かす
  - 変数の置いてあるアドレスが変わる
    ⇒　演算のオペランドを修正する必要
  - 命令の置いてあるアドレスが変わる
    ⇒　ジャンプのオペランドを修正必要



（脱線）命令中のアドレスの書換えはやっかい

- 命令中のオペランド　(覚えていますか)
  - 操作される対象を指定
    例えば、演算命令の対象の変数
    　正確にはメモリ中の変数のアドレス
    例えば、ジャンプ命令の飛び先アドレス
- プログラムのメモリ内の位置を動かす
  - 変数の置いてあるアドレスが変わる
    ⇒　演算のオペランドを修正する必要
  - 命令の置いてあるアドレスが変わる
    ⇒　ジャンプのオペランドを修正必要

で、何が難しいのか？



（脱線）命令中のアドレスの書換えはやっかい

- 命令の位置を見つけるのがやっかい

（脱線）命令中のアドレスの書換えはやっかい

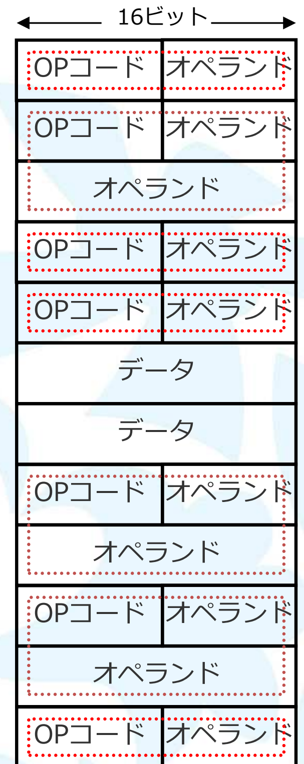
- 命令の位置を見つけるのがやっかい
  - 命令は（普通は）長さがいろいろ

（脱線）命令中のアドレスの書換えはやっかい

- 命令の位置を見つけるのがやっかい

  - 命令は（普通は）長さがいろいろ
    長い命令も短い命令もある
    例：１年生で使ったCOMET IIの命令は
    ２バイト命令と４バイト命令が混在

（脱線）命令中のアドレスの書換えはやっかい



- 命令の位置を見つけるのがやっかい
  - 命令は（普通は）長さがいろいろ
    長い命令も短い命令もある
    例：１年生で使ったCOMET IIの命令は
    ２バイト命令と４バイト命令が混在
  - で、見かけは単なる２進数で、
    OPコードが特殊な形をしていたりしない

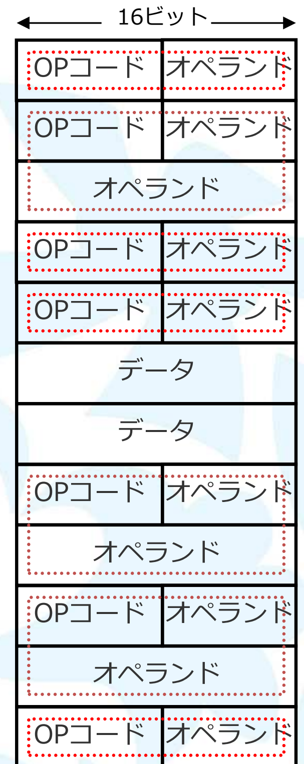
（脱線）命令中のアドレスの書換えはやっかい

- 命令の位置を見つけるのがやっかい
  - 命令は（普通は）長さがいろいろ
    長い命令も短い命令もある
    例：１年生で使ったCOMET IIの命令は
    ２バイト命令と４バイト命令が混在
  - で、見かけは単なる２進数で、
    OPコードが特殊な形をしていたりしない
  - 同じメモリ上にデータも混在してよい

（脱線）命令中のアドレスの書換えはやっかい

- 命令の位置を見つけるのがやっかい
  - 命令は（普通は）長さがいろいろ
    長い命令も短い命令もある
    例：１年生で使ったCOMET IIの命令は
    ２バイト命令と４バイト命令が混在
  - で、見かけは単なる２進数で、
    OPコードが特殊な形をしていたりしない
  - 同じメモリ上にデータも混在してよい

⇒　ぱっと見て命令の区切りは分からない
どこにオペランドがあるのか分からない



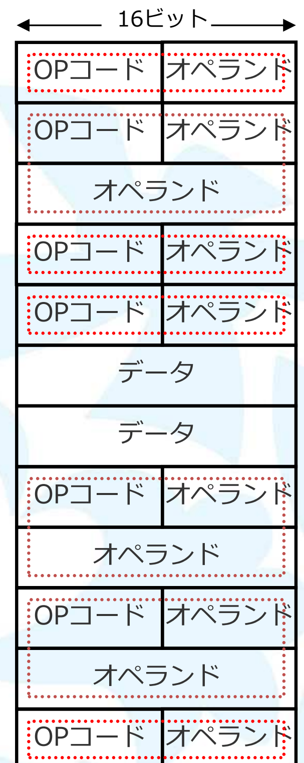
東邦大学

（脱線）命令中のアドレスの書換えはやっかい

- 命令の位置を見つけるのがやっかい
  - 命令は（普通は）長さがいろいろ
    長い命令も短い命令もある
    例：１年生で使ったCOMET IIの命令は
    ２バイト命令と４バイト命令が混在
  - で、見かけは単なる２進数で、
    OPコードが特殊な形をしていたりしない
  - 同じメモリ上にデータも混在してよい

⇒　ぱっと見て命令の区切りは分からない
どこにオペランドがあるのか分からない

見つけるには先頭から実行してみるしかない
現実的でない



東邦大学

ハードウェア支援によるリロケーション

- 命令自身を書き変えるのではなくて

東邦大学

ハードウェア支援によるリロケーション

- 命令自身を書き変えるのではなくて
  - 命令はそのままで
  - CPUから出てからハードで書き直す

東邦大学

ハードウェア支援によるリロケーション

- 命令自身を書き変えるのではなくて
  - 命令はそのままで
  - CPUから出てからハードで書き直す
  - つまりCPU (命令) にとっては
    変換前のアドレスのつもりだが
    メモリ上は変換後のアドレスでアクセス

TOHO UNIVERSITY NATURE LIFE MAN 東邦大学

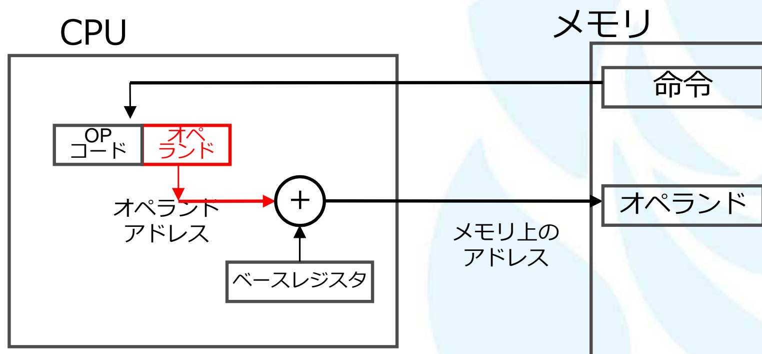
ハードウェア支援によるリロケーション

- 命令自身を書き変えるのではなくて
  - 命令はそのままで
  - CPUから出てからハードで書き直す
  - つまりCPU (命令) にとっては
    変換前のアドレスのつもりだが
    メモリ上は変換後のアドレスでアクセス
    ⇒　ベース（基底）アドレッシング

TOHO UNIVERSITY NATURE LIFE MAN 東邦大学

ハードウェア支援によるリロケーション

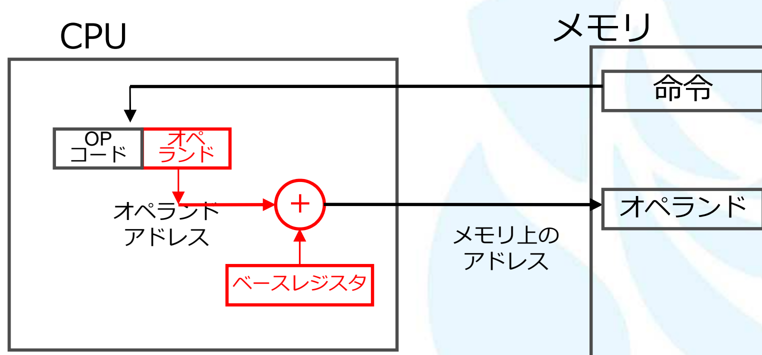
- 命令自身を書き変えるのではなくて
  - 命令はそのままで
  - CPUから出てからハードで書き直す
  - つまりCPU (命令) にとっては
    変換前のアドレスのつもりだが
    メモリ上は変換後のアドレスでアクセス



ベース (基底) アドレッシング
オペランドアドレスに

ハードウェア支援によるリロケーション

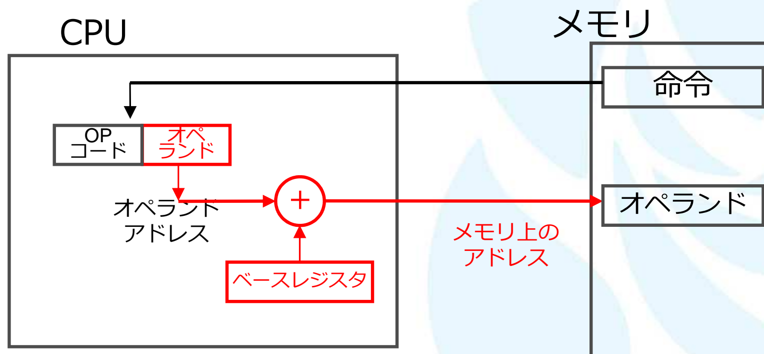
- 命令自身を書き変えるのではなくて
  - 命令はそのままで
  - CPUから出てからハードで書き直す
  - つまりCPU (命令) にとっては
    変換前のアドレスのつもりだが
    メモリ上は変換後のアドレスでアクセス



ベース (基底) アドレッシング
オペランドアドレスに
ベースレジスタ Br を足して

ハードウェア支援によるリロケーション

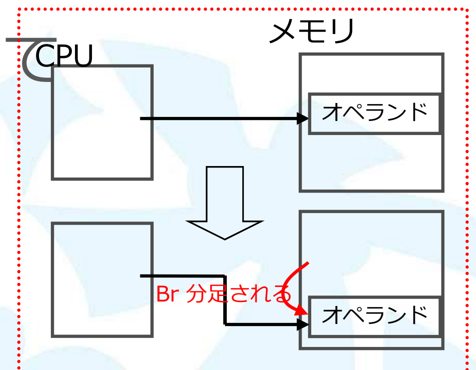
- 命令自身を書き換えるのではなくて
  - 命令はそのままで
  - CPUから出てからハードで書き直す
  - つまりCPU (命令) にとっては
    変換前のアドレスのつもりだが
    メモリ上は変換後のアドレスでアクセス

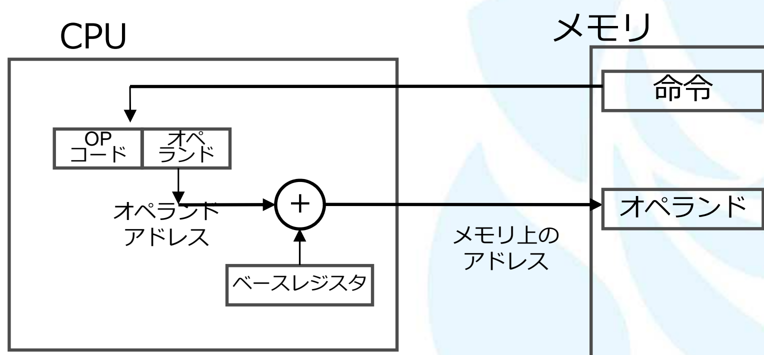


ベース (基底) アドレッシング
オペランドアドレスに
ベースレジスタ Br を足して
メモリへアクセスする

ハードウェア支援によるリロケーション

- 命令自身を書き換えるのではなくて
  - 命令はそのままで
  - CPUから出てからハードで書き直す
  - つまりCPU (命令) にとっては
    変換前のアドレスのつもりだが
    メモリ上は変換後のアドレスでアクセス

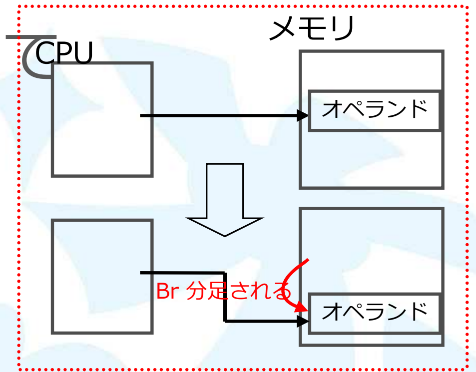


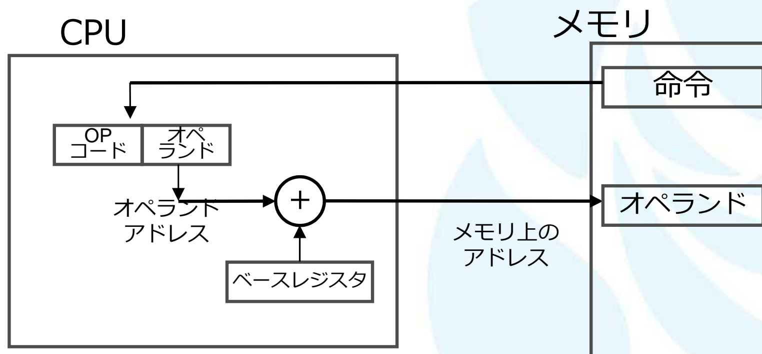


ベース (基底) アドレッシング
オペランドアドレスに
ベースレジスタ Br を足して
メモリへアクセスする
||
Br 分だけずらす変換になる

ハードウェア支援によるリロケーション

- 命令自身を書き変えるのではなくて
  - 命令はそのままで
  - CPUから出てからハードで書き直す
  - つまりCPU (命令) にとっては
    変換前のアドレスのつもりだが
    メモリ上は変換後のアドレスでアクセス





ベース (基底) アドレッシング
オペランドアドレスに
ベースレジスタ Br を足して
メモリへアクセスする
||
Br 分だけずらす変換になる

アドレスを移すという意味でリロケーション（再配置）と呼ばれる

東邦大学

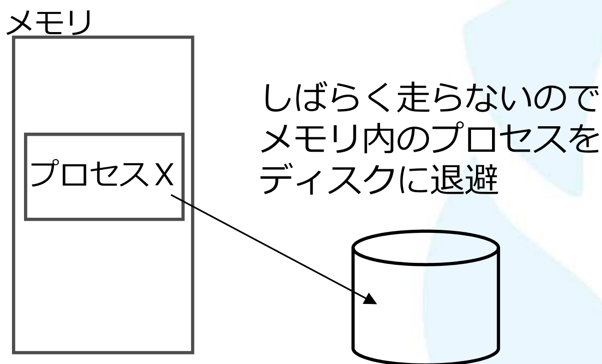
（脱線）プロセス・スワッピングと再配置

- プロセスのスワッピング（入替え）とは

東邦大学

（脱線）プロセス・スワッピングと再配置

- プロセスのスワッピング（入替え）とは
  - 例：　プロセスが事象待ちなどでしばらく走らないことが分かっているとき、メモリから追出して、他のプロセスにメモリを使わせる（中断時のメモリのイメージをディスクに退避）



東邦大学

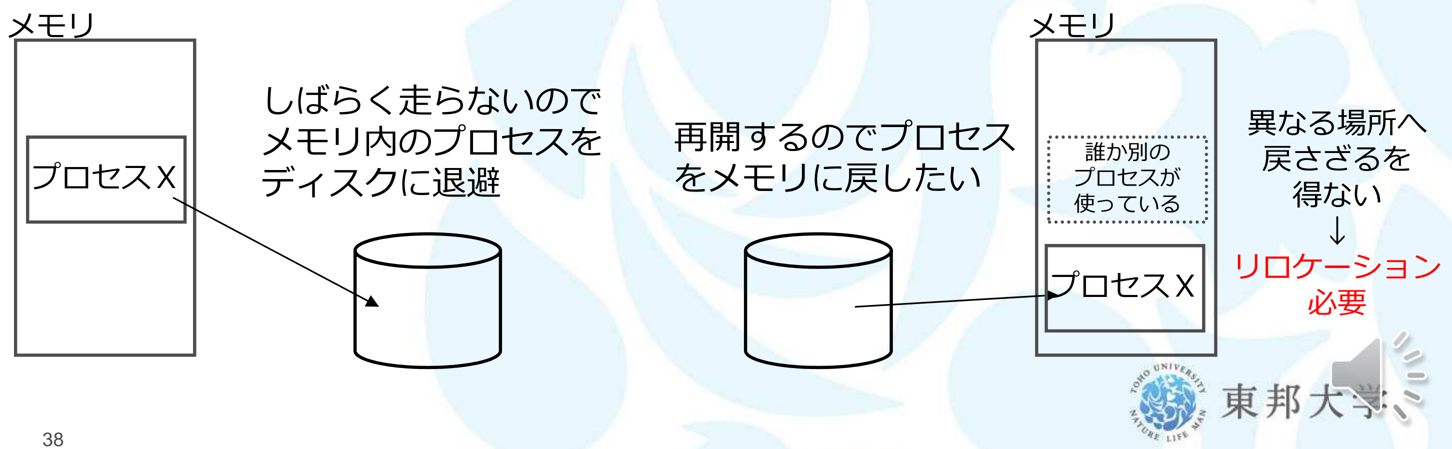
（脱線）プロセス・スワッピングと再配置

- 問題点：　実行に戻すときどうする？
  - 前のときと同じメモリ区画は誰かが使っている違う区画に戻して再開したい

東邦大学

（脱線）プロセス・スワッピングと再配置

- 問題点：　実行に戻すときどうする？
  - 前のときと同じメモリ区画は誰かが使っている
    違う区画に戻して再開したい
    ⇒　リロケーションが必要



東邦大学

ここまでのまとめ

- 固定長割当ては、使えるが
  - 内部フラグメンテーションが起きる
- 可変長割当ては、
  - 無駄は少ないが外部フラグメンテーション起きる

東邦大学

ここまでのまとめ

- 固定長割当ては、使えるが
  - 内部フラグメンテーションが起きる
- 可変長割当ては、
  - 無駄は少ないが外部フラグメンテーション起きる
- プロセスの命令を書き換えて移動するのは、
  - かなり難しくて、現実的ではない
- 基底アドレッシングのハードを使って
  - 移動（リロケーション）が可能
    ⇒　空き地を詰めたりスワッピングもできる

ここまでのまとめ

- というわけで、何とか
  複数プロセスが生成・消滅するシステムの
  メモリの区画管理ができる

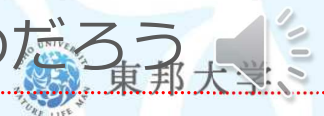
ここまでのまとめ

- というわけで、何とか
  複数プロセスが生成・消滅するシステムのメモリの区画管理ができる
- しかし、
  もう１つの大きな問題があった：
  メモリサイズより大きなプログラムを走らせたい

ここまでのまとめ

- というわけで、何とか
  複数プロセスが生成・消滅するシステムのメモリの区画管理ができる
- しかし、
  もう１つの大きな問題があった：
  メモリサイズより大きなプログラムを走らせたい

メモリが高価だったので、小さな物理メモリでも大きなプロセスを、たくさん同時に（多重に）走らせたい、ということだったのだろう



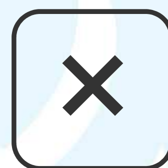
ここまでのまとめ

- というわけで、何とか
  複数プロセスが生成・消滅するシステムの
  メモリの区画管理ができる
- しかし、
  もう１つの大きな問題があった：
  メモリサイズより大きなプログラムを
  走らせたい

⇒　次回の「大きなメモリ」問題

東邦大学

記憶管理の考え方が掴めましたか？



次へ

東邦大学