# オペレーティングシステム
# i386アーキテクチャ(1)

2005年10月21日

海谷 治彦

# 目次

- i386とは
- i386アーキテクチャの内部
- 仮想記憶
  - 実メモリより大きいメモリを扱う
- プロテクトモード
- セグメント

# i386とは

- Intel社のCPU 386以降のアドレスバスが32本あるCPUの総称.
  - だいたいPentium4まで.
- i386, IA32とか80x86とか色々俗称がある.
- 8086, 286等はi386には入らない.
- いわゆるWindowsパソコンに標準的に利用されている.
- AthronとかEfficionとかもソフトウェア側から見れば同じ見える. (互換機)

# i386以外の有名なCPU

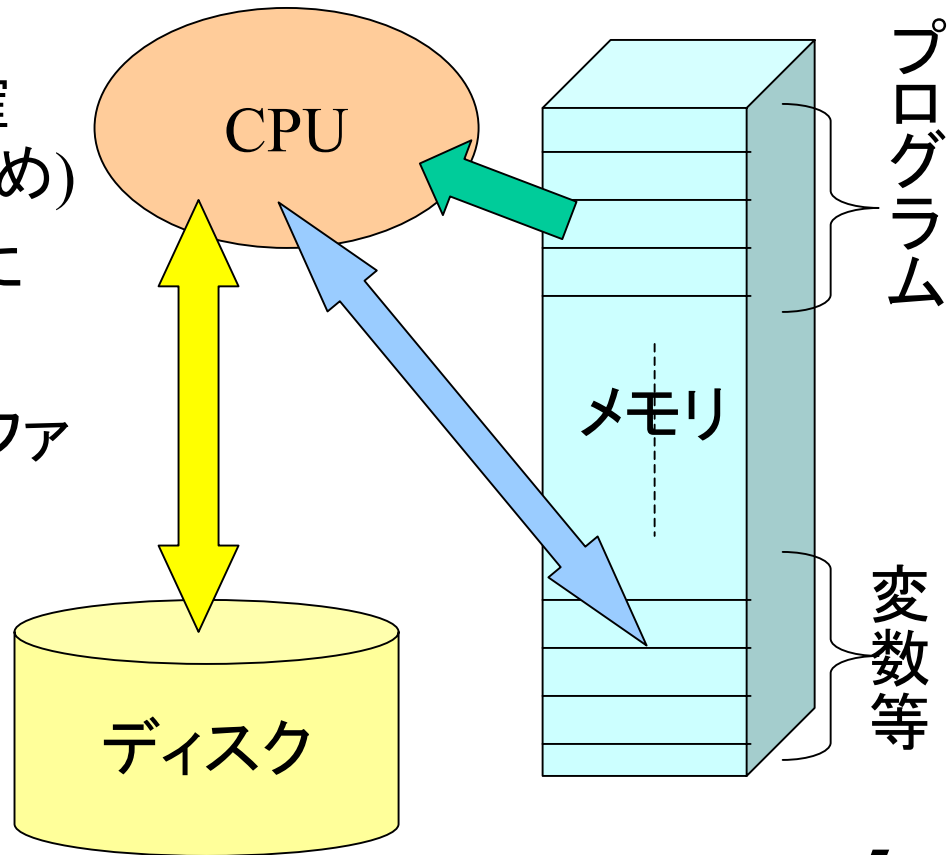
- PowerPC マックで採用
- ARM PDA(ザウルスなんか)に乗っかってた.
- Sparc サンマイクロシステム(Javaの会社)が出してるマシン(いわゆるワークステーション)のCPU.

他，色々ありすぎて列挙不能.

前回より再録: 大雑把なCPU周辺の概念図

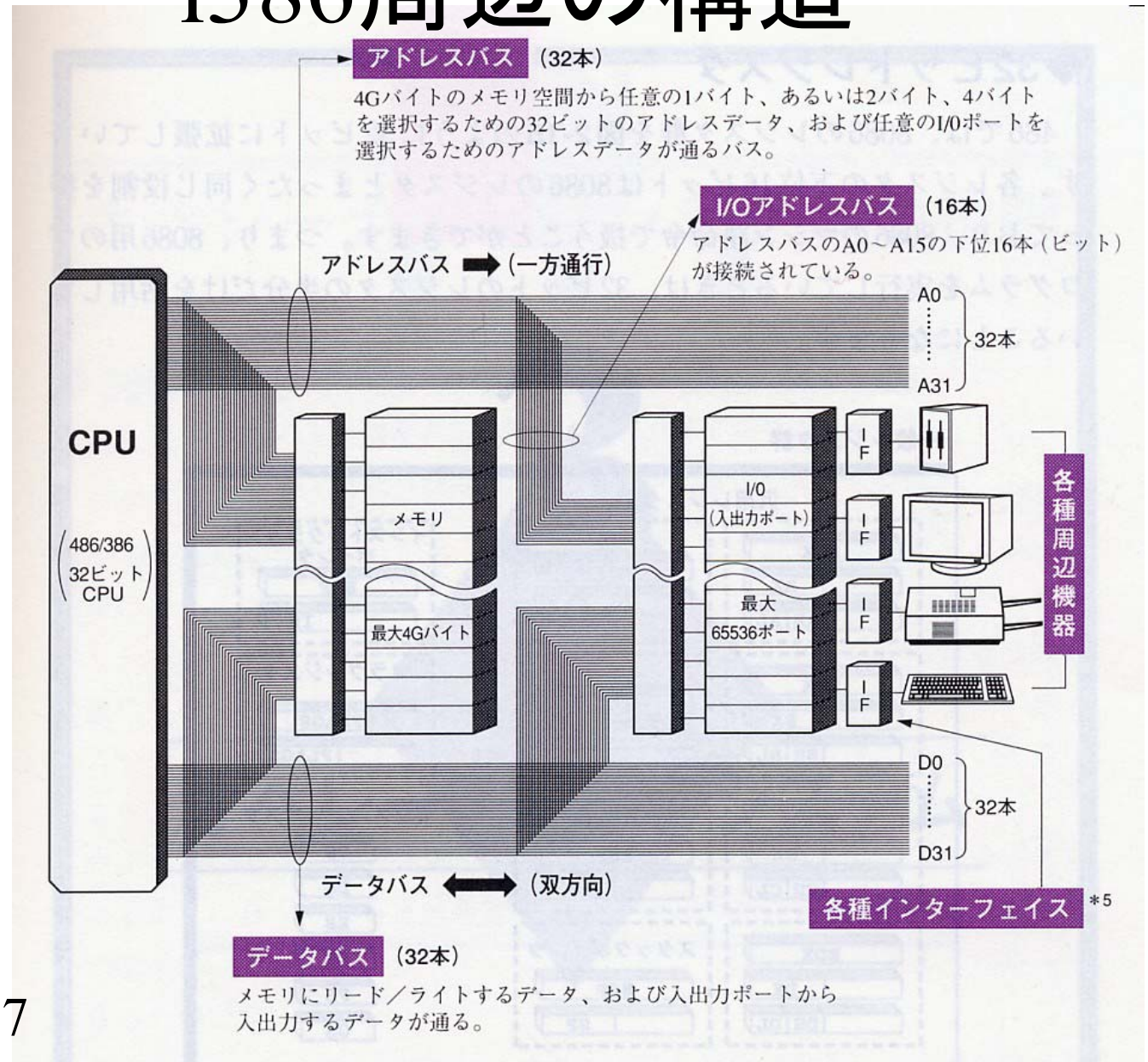
# プログラムの処理の流れ

- プログラムがメモリに読み込まれる.
- 計算に必要なメモリも確保される.（変数等のため)
- CPUがプログラムを順に読んで，計算をする.
- 必要ならば，デバイス(ファイル等)にアクセスする.




アーキテクチャの授業等の復習ですな.

# i386周辺の構造




文献6 p.57

# i386の基本動作

1. アドレスバスでメモリもしくはI/Oポートのアドレスを指定.
2. 指定した場所からデータをレジスタに読む.
3. なんか計算する.
4. またメモリもしくはI/Oポートに結果を書き出す.

の繰り返し.

- プログラム自体もメモリに記録されており，記述される順番に読んで実行するだけ.

# i386のレジスタ

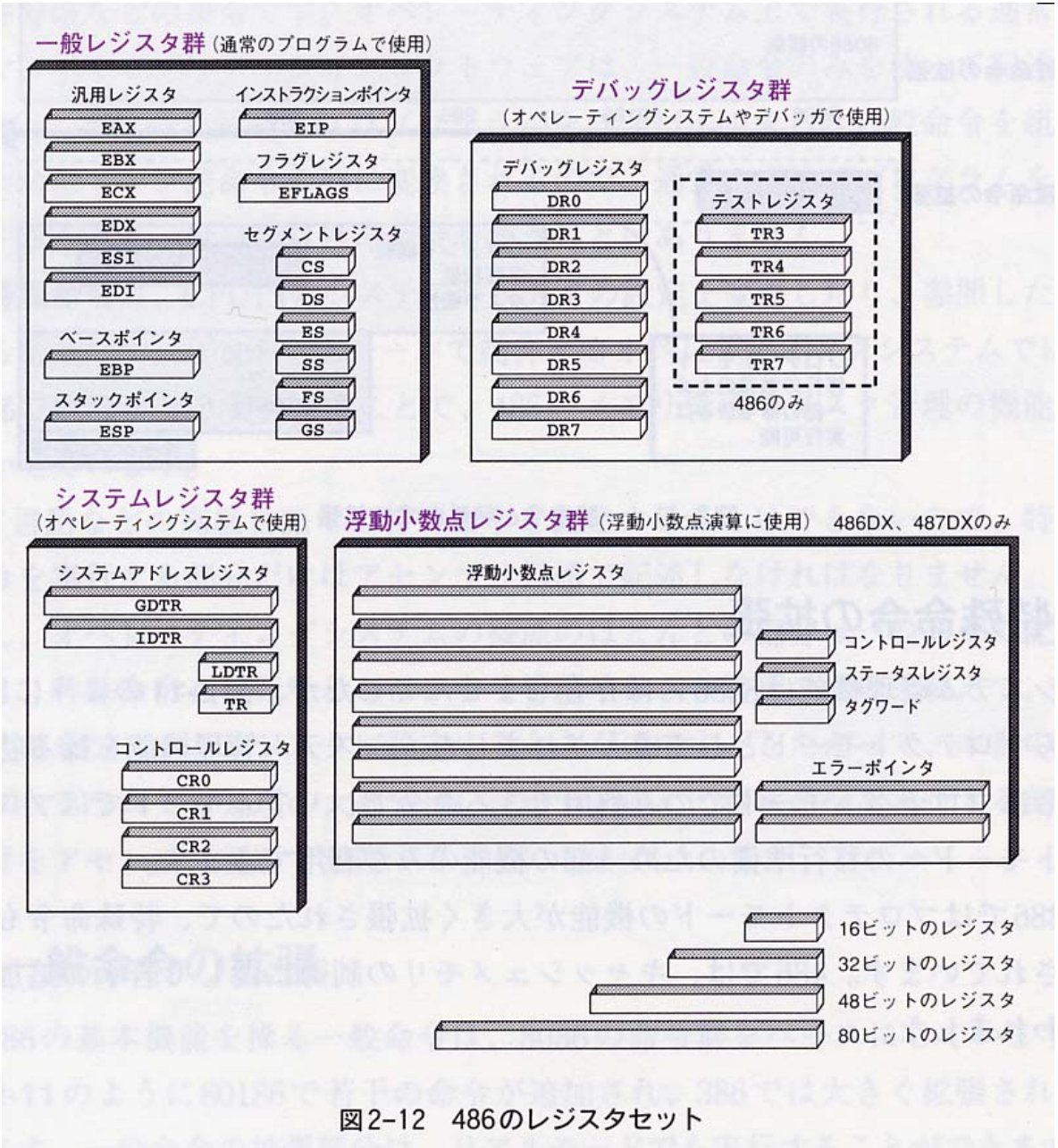


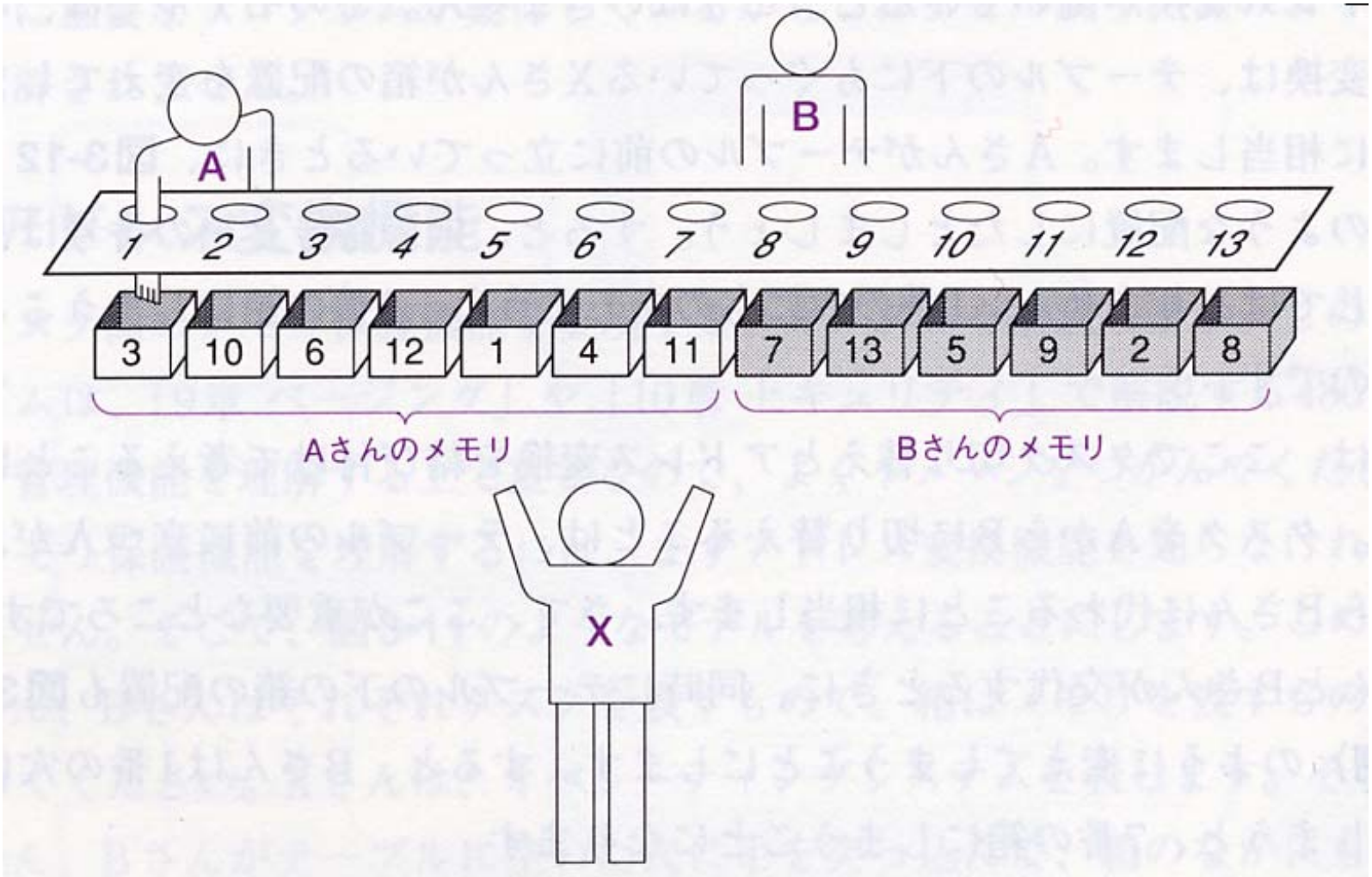

図2-12　486のレジスタセット

文献6 p.52

# プロセスの独立性を保つには？

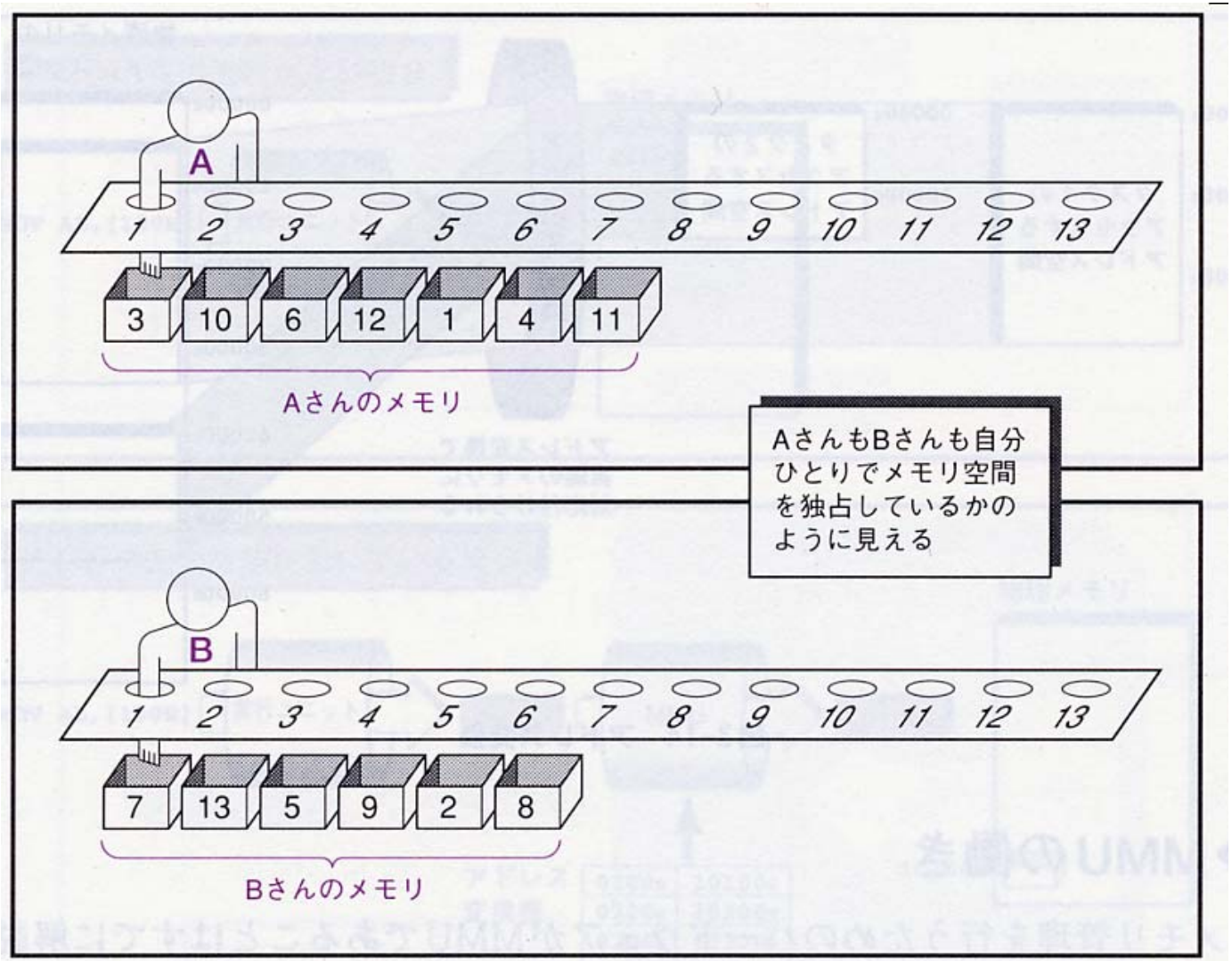
- Linux(UNIX)では，複数のプロセスが同時に存在する.
- 実際のメモリを注意深く区分けして利用するメモリ境界を決めていたのでは危なっかしい.
- そこで，プロセス毎に独立のアドレス空間があるように見せる仕組みが必要.
  - アドレス変換

# アドレス変換の考え方 1




文献6 p.79

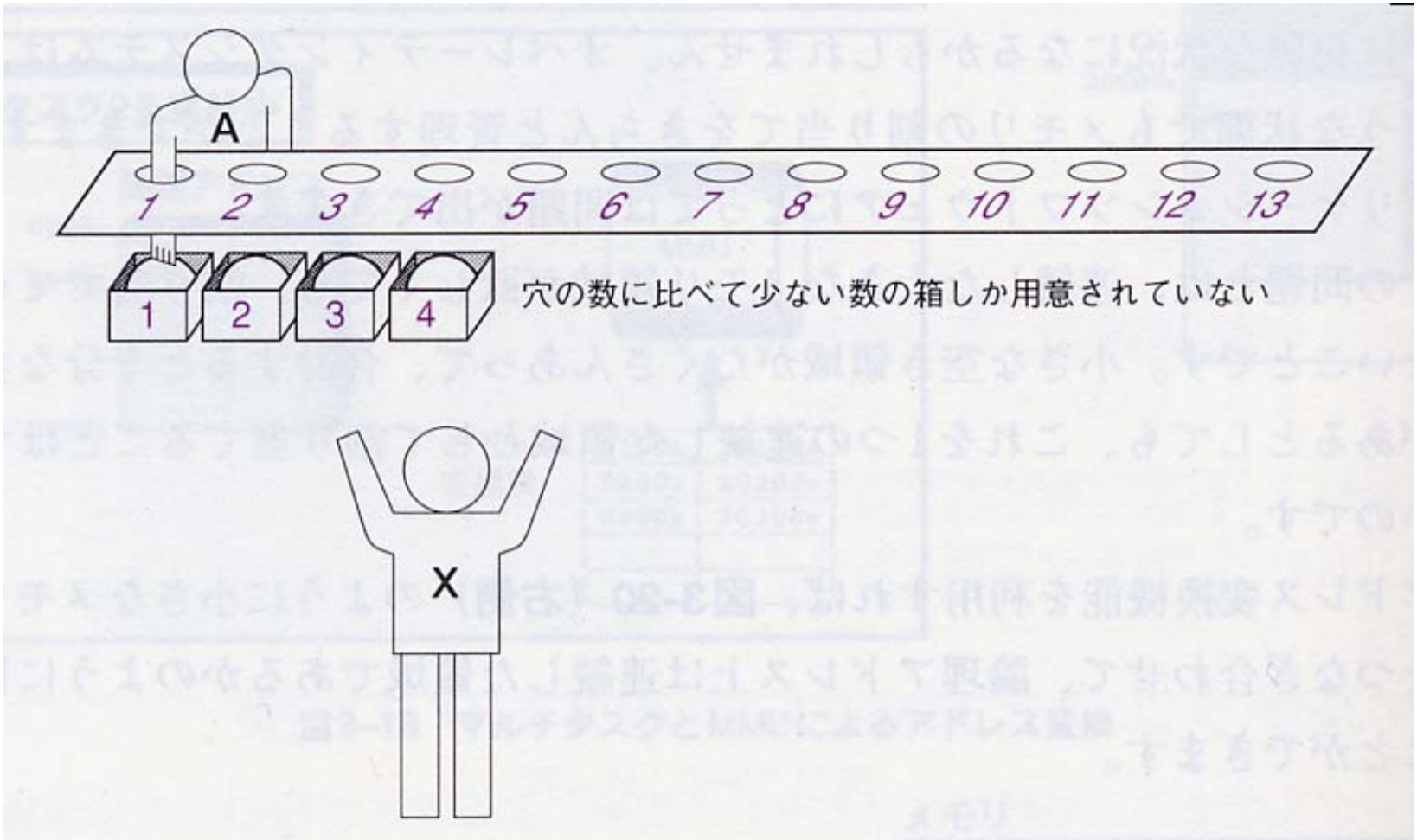
# アドレス変換の考え方 2



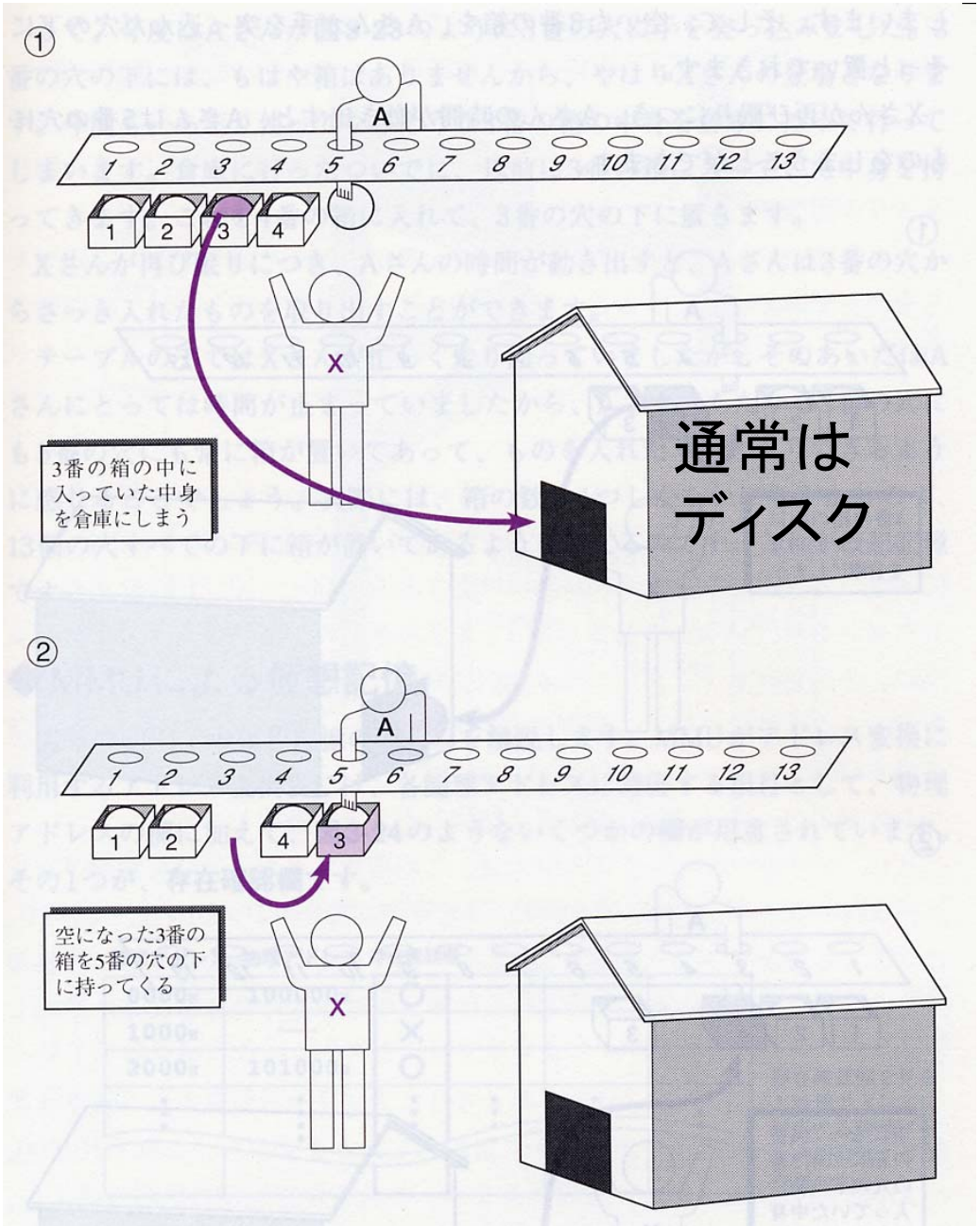
# 少ない実メモリ上で大メモリを

- 実際，4Gのメモリを積んだマシンなどめったにない.
- そこで，実メモリ(512MB程度)を使って，4Gのメモリ空間を表現できなければならない.
  - (実際，4Gまで使うかは別として)
- そのための機構として仮想記憶がある.

# 仮想記憶 1




文献6 p.87




# 仮想記憶 2

左図のようにメモリ内容が，一時退避されることを，「スワップアウト」と言う.

# プロテクトモード

- Protect Mode
- i386が普通(?)に動作している状態.
  - 他のモードについては後日ふれる.
- メモリアドレスが4GB(4294967296=$2^{32}$)
  - 0H～FFFFFFFF H (Fが8個=1が32個)まで.
- プログラムからアドレスを指定する方法が特殊.
- 割り込み(後述)が起こった時の処理が特殊
- メモリ，I/O機器へのアクセス保護がある.
  - プログラムは許可されたアドレス外をアクセスできないようにしている.

# プロテクトモードのための環境

- このモードには複数のプログラム(プロセス)の概念が最初からある.
- よって, これら複数のデータを記録しておくための特別なエリアが必要.
- そのエリアとして,
  - GDT (Global Descriptor Table)
  - CR3 レジスタ (ページ)
  - IDT (Interrupt Descriptor Table)

  があるが詳細は後述.

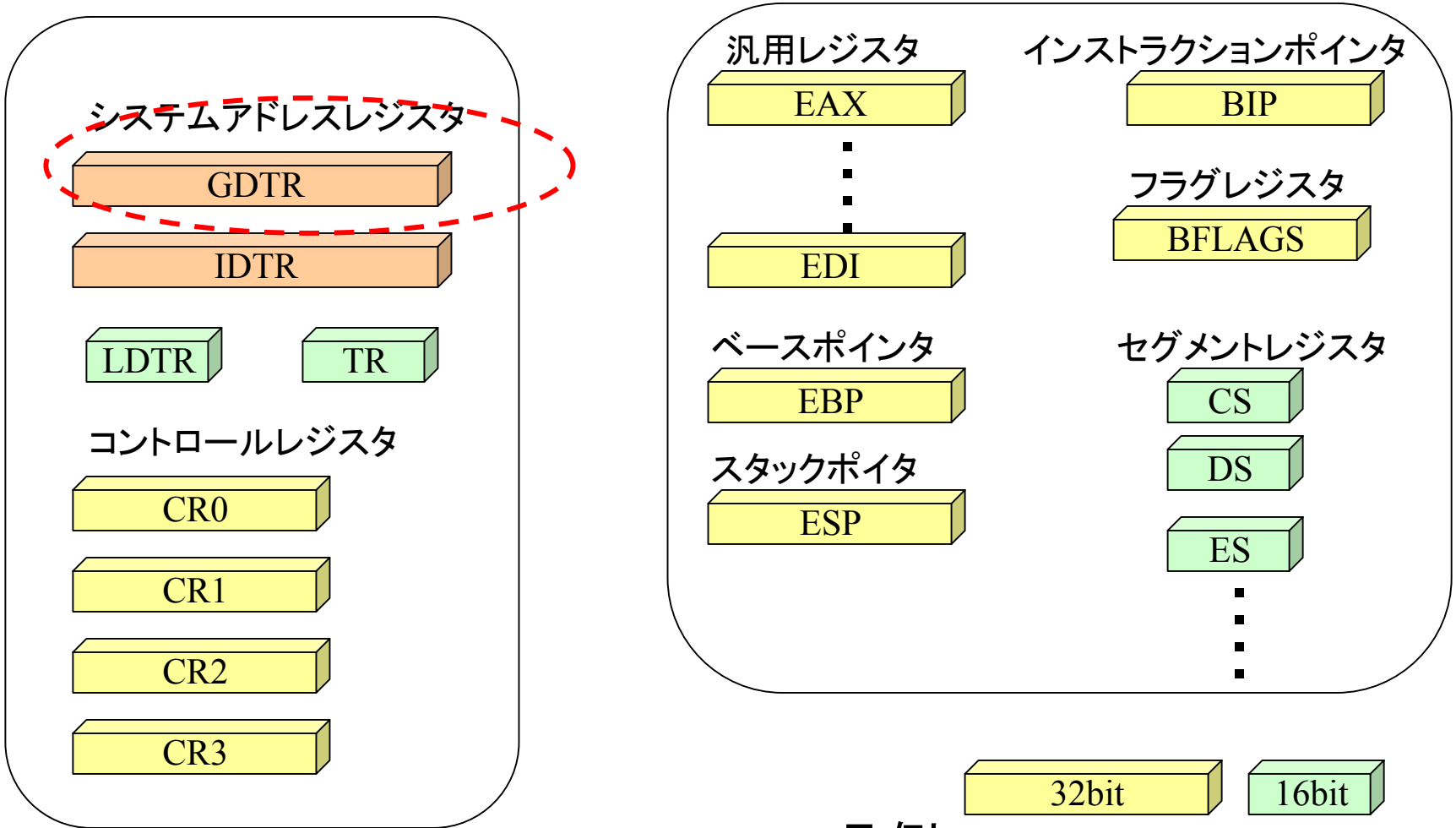
# セグメント

- 4Gのメモリ空間をいくつかの部分に区切る仕組み(および区切りの名前).
  - 重なりがあってもよい.
- セグメント毎に色々情報を設定できる
  - ベース: セグメントの開始アドレス(32bit)
  - リミットとGフラグ: セグメントのサイズを設定
  - タイプ: セグメント内のデータが読み書き可能か，実行可能か等の情報を指定.
  - DPL (Descriptor Privilege Level) セグメント内のプログラムの特権レベルを示す
- 1個分を記述するのに8B必要.

# セグメント群の情報は何処に？

- Segment Descriptor Tableという表に書かれる.
  - Global Descriptor TableとLocal D. T. がある.
- その表はメモリ内に記述しなければならない.
  - 自動的には作られない.
  - 通常，OSが起動時に作成する.
- 表の先頭アドレス(と表内の項目数)は，GDTRレジスタに記述される.
- セグメントレジスタ(CS, DS …)には表の何個目の項目を使うかが記入される.

# i386のOS関係のレジスタ



# Linuxで使うタイプとDPL

- タイプ
  - 1 読み書き可能なデータセグメント
  - 5 実行と読み出し可能コードセグメント
- DPL (Descriptor Privilege Level)
  - レベル0: (最強) どんな命令でも実行でき，どんなアドレスでもアクセスできる.
  - レベル3: (最弱) 特権命令が実行できない.
- i386は8種類のタイプわけ，4種類のDPLレベルがあるが，Linuxでは上記しか使わない.

# Linuxでのセグメント

- 4つのセグメントしか作らない.
  - KERNEL_CS, KERNEL_DS, UESR_CS, USER_DS
- 以下の点が共通
  - セグメントのサイズ 4G
  - セグメントの開始アドレス 0

  要は全部重なってる.
- タイプとDPLについて
  - *_CS タイプ5 (実行可能) 要はプログラムが入る.
  - *_DS タイプ1 (読み書き可能) 要はデータが入る.
  - KERNEL_* レベル0 なんでもできる.
  - UER_* レベル3 機能が制限.

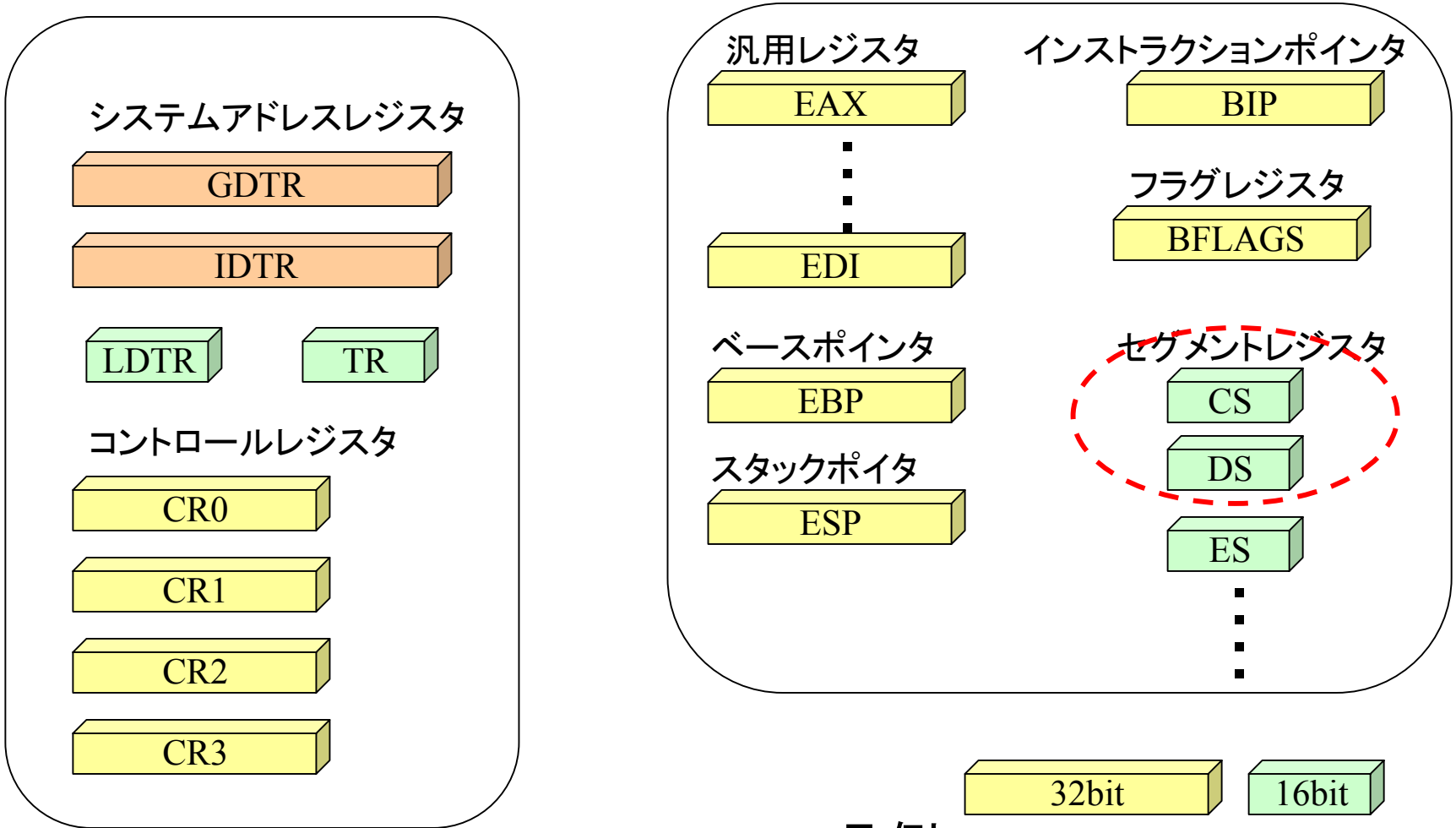
# CSとDS

- CS (Code Segment) プログラムが入っているセグメント
- DS (Data Segment) プログラム実行のための変数等を保存するためのセグメント

- 単純な話,
  - カーネル機能が実行されている場合は, KERNEL_CS KERNEL_DSを利用して計算が行われおり,
  - ユーザープログラムが実行されている場合は, UESR_CSとUSER_DSが利用されている

  ということ.

# i386のOS関係のレジスタ



# Linuxでのセグメント概念図