第10章「コンピュータシステムの高速化技術」

- CPI, IPC
- パイプライン
- 並列処理
- キャッシュメモリ

CPI (Clock Per Instruction), IPC

- CPI = １命令あたり要する（平均）クロック数
- IPC = クロックサイクル辺り実行可能命令数
- プロセッサのスピード＝（クロック周波数）×IPC
- 高速化
  - クロック周波数向上
    - プロセス技術
    - スーパーパイプライン
  - IPC向上
    - パイプライン
    - 並列実行（スーパースケーラ, VLIW）

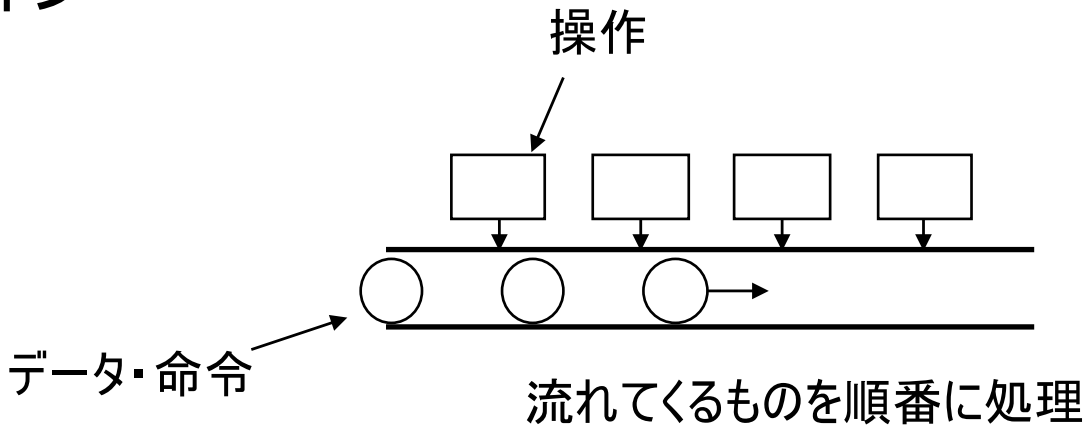
プロセス技術の向上

- 集積度は2（1.5）年に2倍（ムーアの法則）
- 集積度に比例して動作周波数が向上

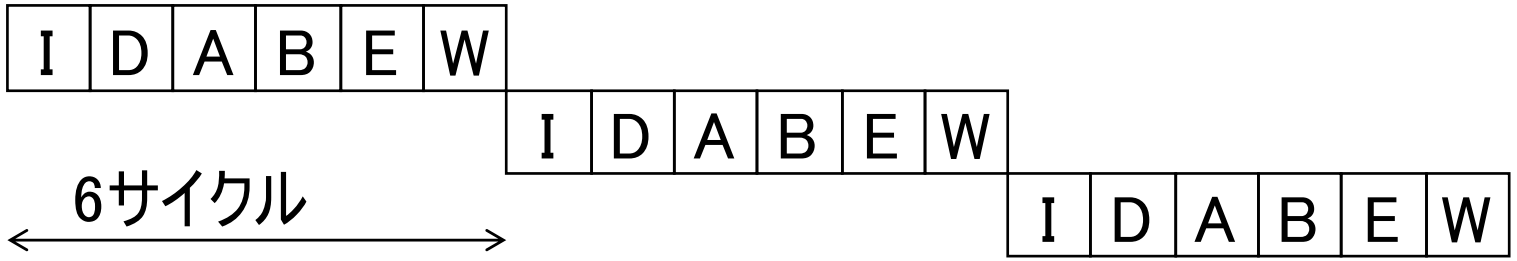
プロセッサ	プロセス	Tr数（×1000個）	動作周波数
4004	10 μm	2.3	–
8080	6 μm	6	2MHz
8086	3 μm	130	5MHz
Intel486	1.0 μm	1200	25MHz
Pentium	0.5 μm	3300	100MHz
Pentium III	0.18 μm	28000	1GHz
Pentium 4	0.13 μm	42000	3GHz

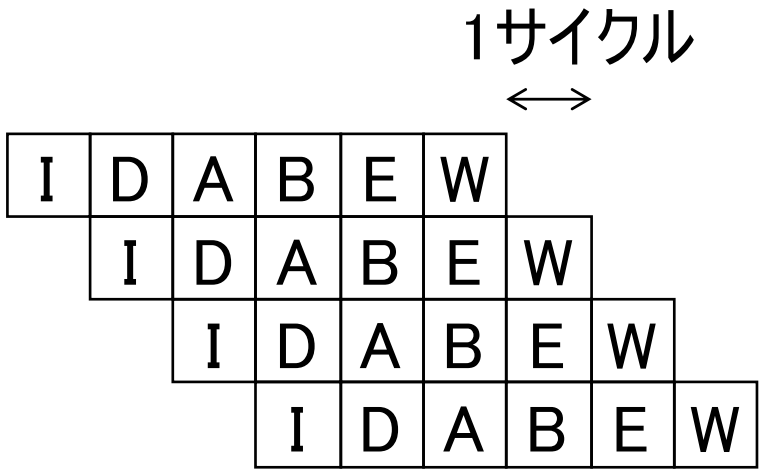
パイプライン

- 命令パイプライン
  - CPUの各回路を遊ばせない
  - 機械命令の実行をオーバーラップさせて高速化
  - 一個の命令あたり平均実行サイクル数（CPI）を減らす
  - １命令平均１サイクルを理想とする
- 演算パイプライン
  - MISD



命令パイプライン





記号	動作内容
I	命令読み出しとPCの変更
D	命令解読
A	アドレス計算
B	オペランドの読み出し
E	命令の演算動作の実行
W	結果の格納

パイプラインハザード

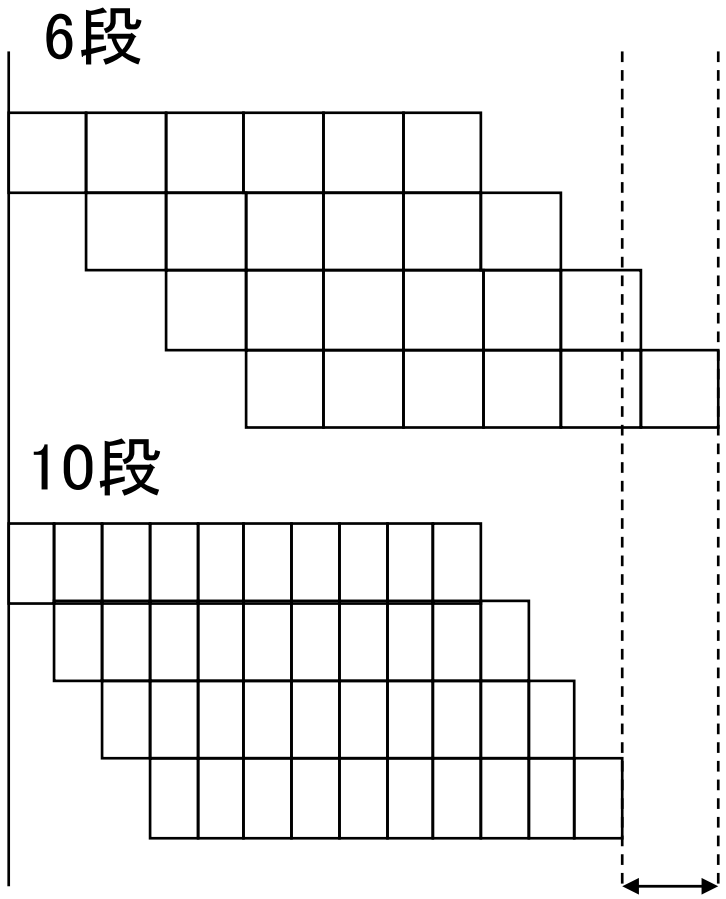
- パイプラインがスムーズに流れなくなること
  - データハザード: 先行する命令の実行結果に依存する
  - 制御ハザード: 分岐命令や割り込みで命令の流れが変わる
  - 構造ハザード: 共通のハードウェア資源に対する競合

RISC (Reduced Instruction Set Computer)

- 設計思想を表す一つの概念
  - CISC (Complex …) と対をなす
  - 例）MIPS, SPARC, PowerPC
- 命令セットを簡略化し，プログラムの高速実行を狙う
  - パイプライン処理を理想的に行う；1つの命令を1サイクルで
  - 制御回路を配線論理方式で
  - コンパイラが重要に；パイプラインがスムーズに流れるように自動的にコードを生成

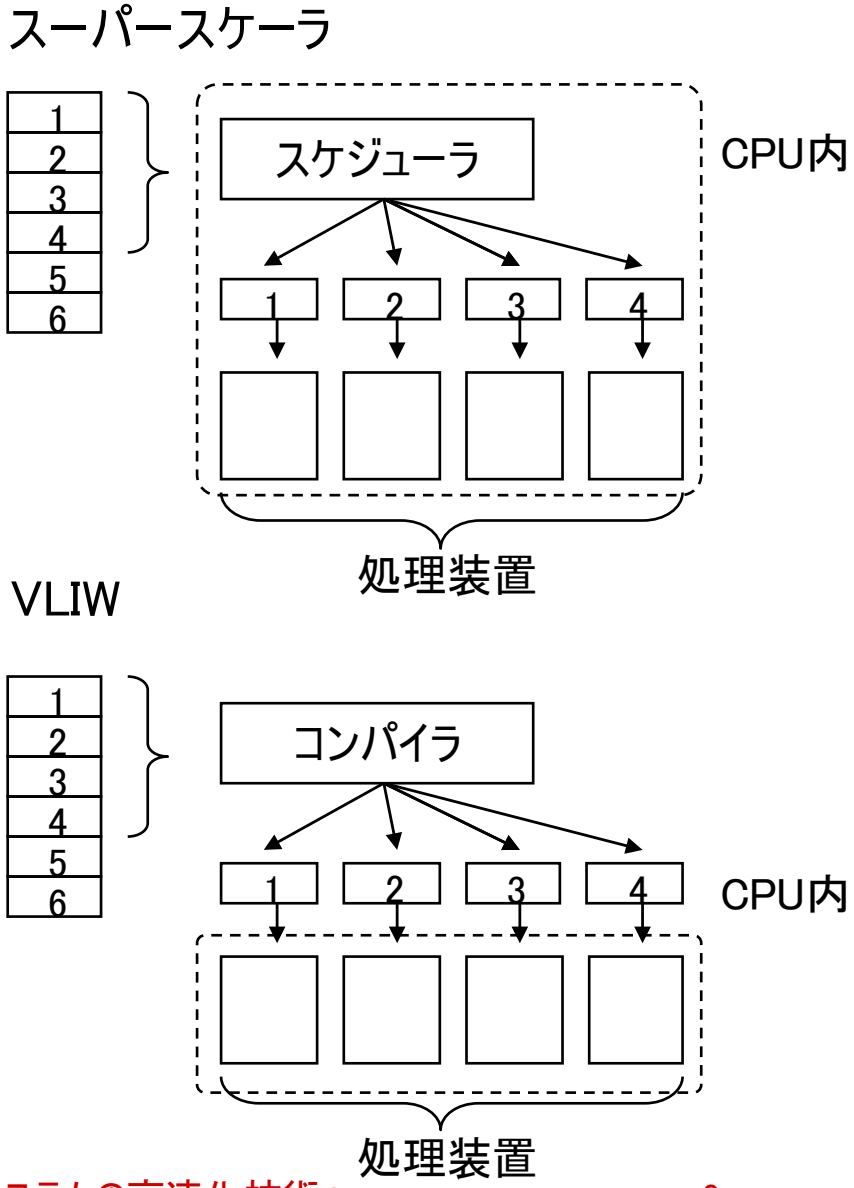
スーパーパイプライン

- パイプラインの段数を増やし，より細かいステージに分割すること
- 各ステージの処理が単純になり，回路が単純に→クロック周波数を上げられる
  - PentiumIII: 段数12, 1GHz
  - Pentium4: 段数20, 3GHz
- ハザード時のペナルティはそれだけ大きくなる弱点も



並列実行

- Flynnによる並列計算の分類
  - SISD (Single Instruction Single Data): 逐次計算機
  - SIMD : DSP, 音声画像処理
  - MIMD : マルチプロセッサの並列計算機
- CPU内に複数の処理装置を置き，命令を同時に実行する
  - スーパースケーラ
  - VLIW
- マルチコアCPU
  - MIMDの一種
  - Intel Core 2



スーパースケーラ(Super Scalar)

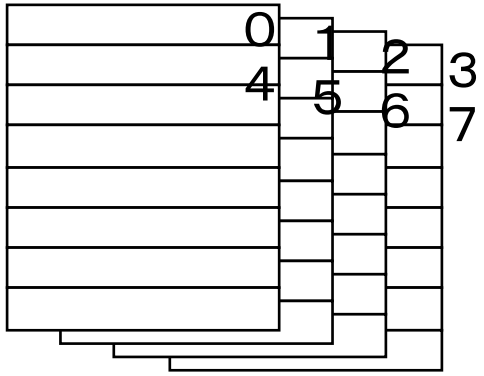
- 同時に複数の異なる命令を実行する
  - 複数のパイプラインを持つ
- 命令には依存関係があり，いつも同時に実行できるとは限らない
  - 投機実行:分岐予測，
- 例： 同時命令実行可能最大数：Pentium4(6), AthlonXP(9)

VLIW（Very Long Instruction Word）

- 依存関係にない複数の命令をまとめて一語とし，CPU内の同数の演算処理装置で並列実行
- プログラムの都合で依存関係にない命令を複数用意できない場合は，なにもしない（NOP）命令を組み合わせる
- ハードウェア側ではスケジューリングをしないので簡素にできる
- コンパイラが依存関係などの判断を行う
- 例）
  - Transmeta, Crusoe

メモリの高速化

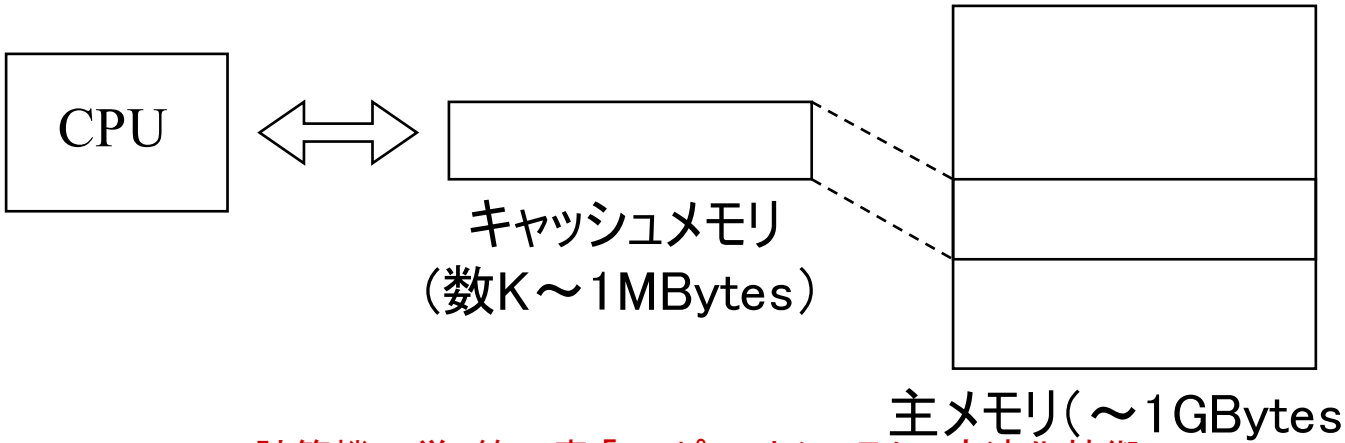
- CPUの速度に比べてメモリは非常に低速
- 高速化
  - アクセス速度の向上
    - バーストアクセスの高速化（SDRAM etc.）
    - インターリーブ:異なるメモリモジュールにアドレスを割り振る
  - キャッシュメモリ
    - 高速小容量のメモリを間に挟む
    - メモリだけでなくハードディスクも同じ



メモリインターリーブ

キャッシュメモリ（cache memory）

- プログラムとデータの時間的・空間的局所性に着目し，小容量だが高速なメモリをCPUと主メモリの間に挟んで，実効的アクセス速度を向上
  - 時間的局所性：一度参照されたプログラムまたはデータの語は近い将来再度参照される可能性が高い
  - 空間的局所性：一度参照されたプログラムまたはデータの語の近くの語は近い将来参照される可能性が高い



キャッシュメモリの動作

- キャッシュメモリには主メモリの一部を格納
- キャッシュメモリ上に欲しいデータがあればキャッシュメモリにアクセスし，なければ（キャッシュミスと言う）主メモリから内容をコピーする
- 通常，キャッシュメモリは分割して管理され，キャッシュミス時，一番最後にアクセスされた部分を入れ替える（Least Recently Used;LRU）
- 書き込み時，キャッシュと主メモリ間でのデータの一致（coherency）を保つ必要がある
  - ライトスルー方式：書き込み時主メモリにも書き込む
  - ライトバック方式：キャッシュ入れ替え時に主メモリに戻す

キャッシュメモリの実効アクセス時間

- キャッシュミスがあるので，実効的なアクセス時間は次のように計算される

$$T = h \cdot T_C + (1-h) \cdot T_m$$

ヒット率　　キャッシュメモリアクセス時間　　主メモリアクセス時間

例）h=0.9, T_c=1, T_m=10 なら T=0.9+0.1 × 10=1.9

キャッシュメモリの使い方

- キャッシュメモリの効果は非常に大きい
- 頻繁にキャッシュメモリの内容が書き換えられる状況では処理速度は落ちる
- OSやアプリケーションプログラムに依存する

```
int array[M][N];
for (i = 0; i < M; i++) {
  for (j = 0; j < N; j++) {
    x = array[i][j];
    ...
  }
}
```

```
int array[M][N];
for (j = 0; j < N; j++) {
  for (i = 0; i < M; i++) {
    x = array[i][j];
    ...
  }
}
```

左のプログラムの方が一般に高速