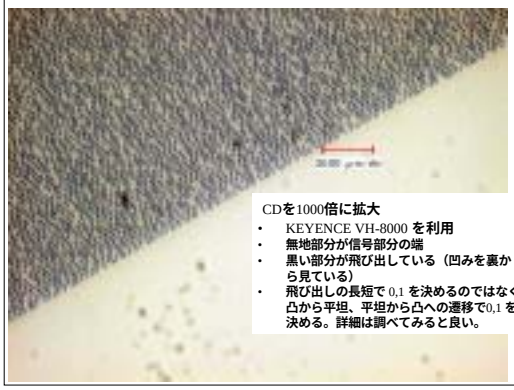
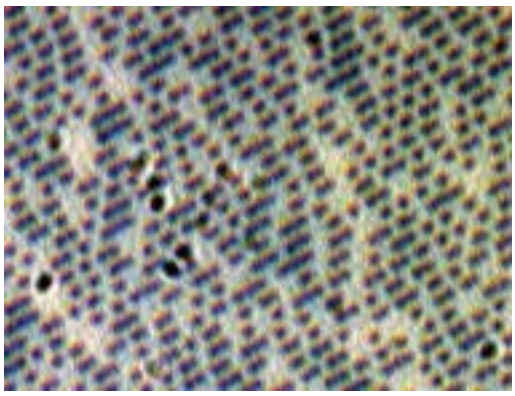
顕微鏡写真



CDを1000倍に拡大
- KEYENCE VH-8000 を利用
- 無地部分が信号部分の端
- 黒い部分が飛び出している（凹みを裏から見ている）
- 飛び出しの長短で 0,1 を決めるのではなく、凸から平坦、平坦から凸への遷移で0,1 を決める。詳細は調べてみると良い。



情報処理の概念

#7 性能、複雑さ、バグ、トレードオフ / 2002 (春)

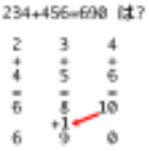
一般教育研究センター 安田豊

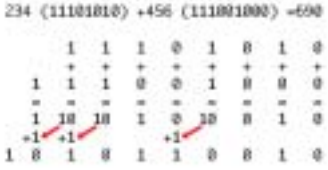
繰り返し処理

- (一般的) コンピュータの特長
  - 単純な処理しかできない装置を組み合わせ、繰り返して処理することで複雑な処理をこなす
    - 前提：「複雑な処理」は単純な処理に分解可能である
  - コンピュータの処理対象の限界をも示す
    - 単純な処理に分解できない仕事には対応できない
    - 多くの場合、分解できない＝よく分かっていない
    - つい先日まで二足歩行ができなかった
- 繰り返し処理の例
  - 二進での多数桁の足し算・掛け算

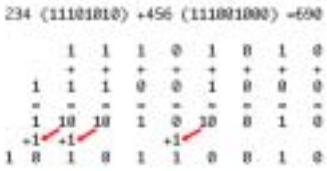
二進での計算

- 10進で3桁の足し算を分解
  - 10進1桁の足し算を三回(繰り上がり込み)
- 2進では9桁、足し算も9回





二進での計算



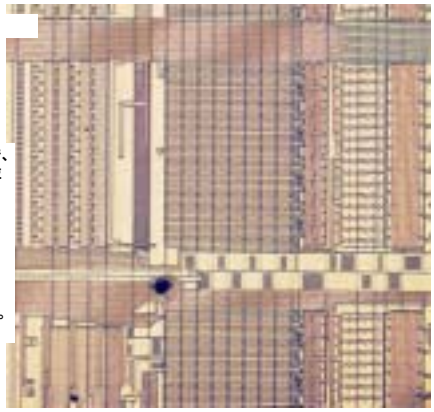
- このような方法(筆算)で処理を行なう場合、
  1. 一つの素子を9回使い回して処理する
  2. 素子を9つ並べて一回で処理するか

  のいずれかとなる。
- 実際のコンピュータ (32bit CPUの場合)
  - 32桁を一度に計算して、
  - 32桁以上の精度が必要な場合は何度も繰り返す

性能 (処理速度) は何で決まるか

- 並列度（例えば一度に処理する桁数）
  - 4bit CPU では (世界で初めての CPU Intel 4004, 1971)
    - 4桁単位で処理
    - 回路を 4 並列で用意して実現
    - 4桁以上の演算は繰り返しで処理
  - その後 8bit, 16bit, 32bit, 64bit が処理能力のために開発
  - 現在市場での高性能CPUは 64bit CPU が主流
- なぜ徐々に上がるのか？もっと上がらないか？
  - 性能＝回路の複雑さ
  - 性能向上は、技術的困難さと価格の問題に直結する

CPU回路の拡大



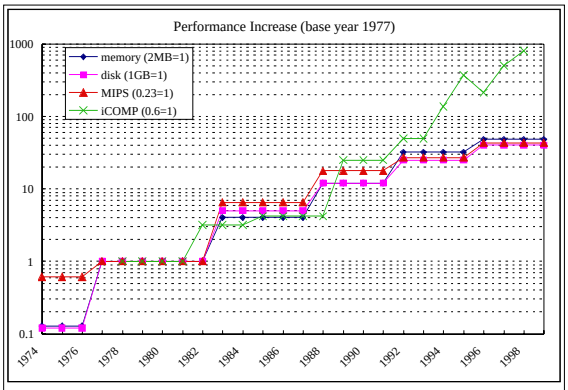
64bit CPUなので、同一回路が64 並列で並ぶ。
目に映る回路は 16 並列なので、恐らくこの見えている構造の中に 4 つずつ何かの回路ができていると思われる。

資料

資料2

性能 (処理速度) は何で決まるか

- 繰り返し周期
  - Intel 4004の108KHzから、今では 2GHz 程度に
  - 性能＝高速度
  - 電気の伝わる速度、素子が機能する最短時間との闘い
  - 再び技術的困難さと価格の問題に直結
- 「単純な処理の高速な繰り返し」で性能が決まる
  - この方法での高速化の限界がある
  - ブレイクスルーが望まれる
- 参考：
  - 「マイクロプロセッサのテクノロジ」Intel の Web
  - http://www.intel.co.jp/jp/home/technology/processor/index.htm



Yutaka Yasuda, 1999, RIEB of Kobe University

バグ (復習)

- プログラムに含まれる「間違い」
  - データは意味をもたない
    - 人間だけが知っている
  - コンピュータは意味を扱わない
    - コンピュータに「間違い」「正しい」という概念はない
  - 無意味な（矛盾した）処理をプログラムしても指示通り動作する
    - 例えば「金利と残高を加算する」ところを間違えて「年齢と金額を加算」させてしまうかもしれない
    - 結果からどこを間違えたのか調べて修正するのが非常に困難
- より複雑な処理のために
  - ブレイクスルーが求められている
  - ソフトウェア工学などを見よ

ソフトウェアの複雑さ

- プログラムの例
  - 「1から10までの数を足した結果を得る」
  - これはコンピュータにとって「難しすぎる」指示
    - コンピュータは頼まれた計算処理をするだけ
    - 「何をするか」では一歩も進めない
    - 「どうやるのか」を明記して与える必要がある
    - 明確な計算の手続きで仕事の内容を記述しなければならない
- プログラムとは何か
  - 目的に対して「どうやるのか」を記述したものがプログラムであり、その記述作業がプログラミングである。
  - 日本語的プログラム
    Xを 1 から 10 まで変化させ、それを毎回 Y に繰り込む
  - コンピュータはデータの意味を理解しないのと同様に、プログラムの意味も知らない（手順だけを知っている）

ソフトウェアの複雑さ

- 「Xを 1 から 10 まで変化させ、それを毎回 Y に繰り込む」のもコンピュータには複雑すぎる
  - 彼は日本語を理解できない
  - コンピュータが理解できる言語で書きくだす必要がある
- C言語

```
j=0;
for(i=1;i<=10;i++) {
  j=j+i;
};
```

  - jははじめ 0 だと設定している
  - 1から10まで変化させるということを、「1 からはじめて 10 以下の場合は終わりまでの処理を行い、1 加算してもう一度繰り返し」という表現で明記している

ソフトウェアの複雑さ

- しかしC言語でもコンピュータのハードウェアは直接理解できない
  - さらに単純な処理に分解しなければ
  - CPU(例は Motorola 68000 を仮定)向けのアセンブラで記述

```
        st 0,$1004
        mov 1,%o0
        st %o0,$1000
.LL2:
        ld $1000,%o0
        cmp %o0,11
        bgt .LL3
        ld $1004,%o0
        ld $1000,%o1
        add %o0,%o1,%o0
        st %o0,$1004
        ld $1000,%o1
        add %o1,1,%o0
        mov %o0,%o1
        st %o1,$1000
        b .LL2
.LL3
```

元の記述

```
j=0;
for(i=1;i<=10;i++) {
  j=j+i;
};
```

ソフトウェアの複雑さ

- アセンブラでもまだ直接は理解できない。
  - 更に機械向けの言語(機械語)に直さなくては
    - アセンブラー行が数バイトの機械語に変換される
  - （例は機械語の一部）

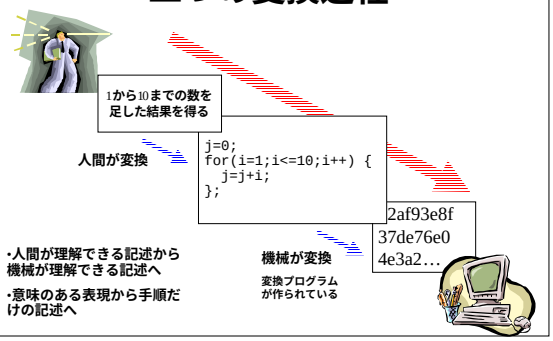
```
7400 5f5f 4354 4f52 5f4c 4953 545f 5f00
5f65 6e76 6972 6f6e 005f 656e 6400 5f47
4c4f 4241 4c5f 4f46 4653 4554 5f54 4142
4c45 5f00 6174 6578 6974 0065 7869 7400
....
```

- 日本語で一行だった処理が大量の手順列に変換された

二つの変換過程

- ソフトウェアに存在する二種類の変換
  - 目的から手順列への変換
    - 人間が行う
    - この作業をプログラミングと呼ぶ
    - コンピュータが理解できる言語で記述
  - プログラミング言語から機械語への変換
    - 極めて限定的なハードウェアの機能に適合させるための作業
    - （大抵の場合）機械が行う
    - この変換を行うのもプログラムである

二つの変換過程



失われれる意味

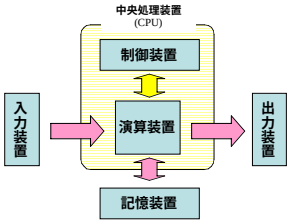
- 目的から手順列への変換過程で、プログラムから意味は失われる
  - そのプログラムの目的を知るのはプログラマだけ
- コンピュータは情報から意味を抜いたデータだけを扱う
  - その意味は入出力の前後にいる人間だけが知る
- ソフトウェアは作業から意味を抜いた手順だけを扱う
  - その意味は結果を受け取る人間だけが知る
- バグ（意図しないプログラムの振る舞いをひきおこす不具合）が発生する根本的要因

プログラミング言語から機械語への変換

- ハードウェアが理解できる命令列は極端に単純な機械語
  - せいぜい何ビットかの加減算程度
  - 乗算、除算ができる場合もある
- 処理装置内に数値を一時置きできるのも僅か
- 複雑な作業は（1から10までの加算ですら）
  - 演算装置とその周囲の間を、データを「とっかえひっかえ」しながら処理するしかない

プログラミング言語から機械語への変換

- プログラムとはつまりハードウェアの「やりくり」指示
  - 実際の処理もひたすらデータのやりくりに尽きる
  - 演算装置と周辺との間でデータの交換が激しく行われることが想像できるだろうか？
- 性能（処理速度）が重要な理由
  - 日本語では数行で済む処理を、機械語では大量の手続きと長い繰り返し作業で実現



トレードオフ （両立しない競合要素）

- 性能向上のために
  - ハードウェアを高機能にすれば無駄な「やりくり」が減って処理能力があがるのでは？
  - ソフトウェアの変換も簡単になるのでは？
    （例えばC言語を直接実行するCPUを作る、など）
- トレードオフ
  - どこまでをハードウェアにやらせて、どこからをソフトウェアで処理すべきか？
  - プロセッサ（処理装置）を複雑にすると回路が複雑になって処理速度があげられない、価格も上がる
  - 逆に単純にしすぎるとソフトウェア処理が煩雑になる
  - そのときの技術の進度に応じて着地点が決まる
  - これが情報処理システムの設計そのものである
    - Java Chip, VLIW などトライアルも多く行われている