計算機システム概論 II

第2回（平成１５年度９月１２日）

コンピュータ（CPU)の仕組みと CPUの高速化

担当　佐藤

計算機システム概論 II

もくじ

- “CPUは何をしているのか”
  - トランジスタ、論理回路からマイコンまで
- CPUを早くする方法
  - パイプライン
  - スーパースカラ
  - …
- 次回、並列処理、グリッドコンピューティング

いろいろなマイクロプロセッサ（1）

- マイコン（４ビットマイコン）
  - ４００４(世界初、1971年、750KHz)
- ８ビットマイコン
  - ８００８(1972年、500KHz、インテル)
  - ８０８０(1974年、2MHz、インテル)
  - ｚ８０(1976年、10?MHz、ザイログ)
  - MC6800（1974年、1MHz、モトローラ）
  - MC6809
- 16ビットマイコン
  - ８０８６（1978年、インテル）
    - IBM PC/MS-DOS
  - ８０２８６(1982年、インテル）
  - MC68000 （1979年、モトローラ）
    - UNIX
- ８ビット、１６ビットとは、バスの幅、メモリ空間のビット幅のこと。

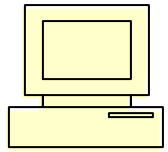
計算機システム概論 II

いろいろなマイクロプロセッサ（2）

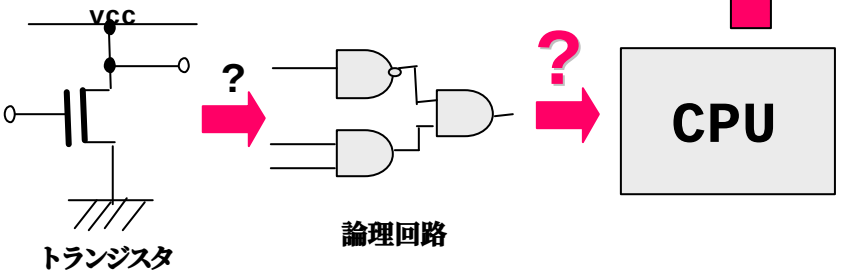
- ３２ビットプロセッサ
  - 80386 (1985年）、80486 (1989年、40MHz～)
  - MC68020(1984年）、MC68030 (1987年)
    - 仮想記憶
  - Pentium (1995年,100MHz～200MHz)
  - Pentium II (1998年,300MHz～)
    - SSE/MMX
  - Pentium III (1997年、900MHz～)
    - 1GHzを超える
  - Pentium 4 (2000年、～3.2GHz)
  - AMD K9, AMD Athlon
- 64ビットプロセッサ
  - Itanium(2000), Itanium II (2001)
  - AMD Opteron (2003)
- ３０年間で、1MHzから1GHz、1000倍の進歩

CPUと論理回路

- CPUは、論理回路である！
- では、どうやってうごいているのか？

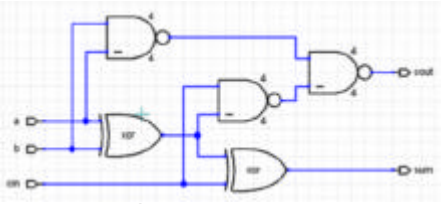


コンピュータシステム

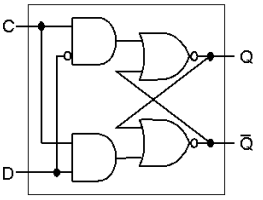


論理回路

- 組み合わせ回路
  - 入力により、出力が決まる
  - 論理素子AND, OR,NOTの組み合わせ
  - 加算器
- 順序回路
  - 現在の出力が過去の入力の状態によって決まる回路
  - フリップフロップ
  - レジスタ、カウンタ、…
  - メモリ（電荷素子）



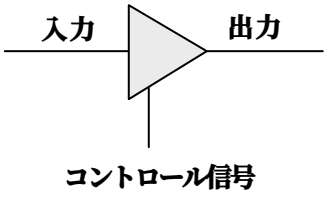
full adder

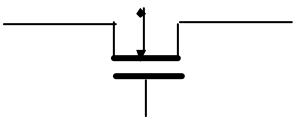


Dラッチ

論理回路

- バススイッチ（ゲート）
  - コントロール信号によって、出力を遮断する。





コンピュータの基本的な構成

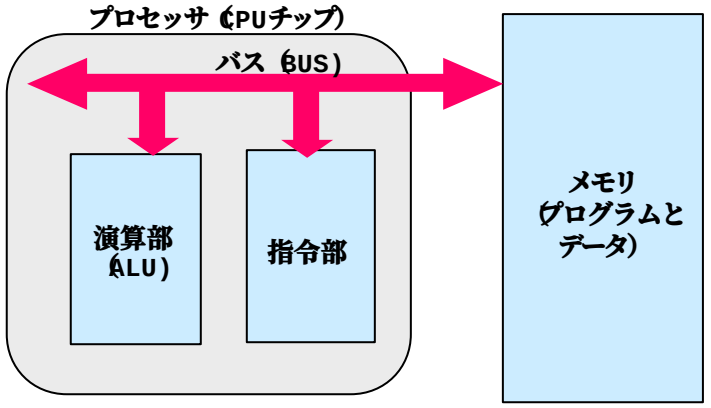
- コンピュータのもっとも基本的な要素は、メモリとプロセッサ（CPU)である。
  - メモリはプログラムやデータを格納する場所
  - プロセッサはそのメモリからプログラムやデータを読み出して、プログラムを実行しています。
    - 指令する部分：プログラムを解釈（？）して指令する
    - 演算する部分：足し算や掛け算をする部分
- プログラムとデータをメモリにいれて、プロセッサがメモリから読み出して実行する方式を、ストアードプログラム方式という

現在のコンピュータのもっとも重要な基本的な概念

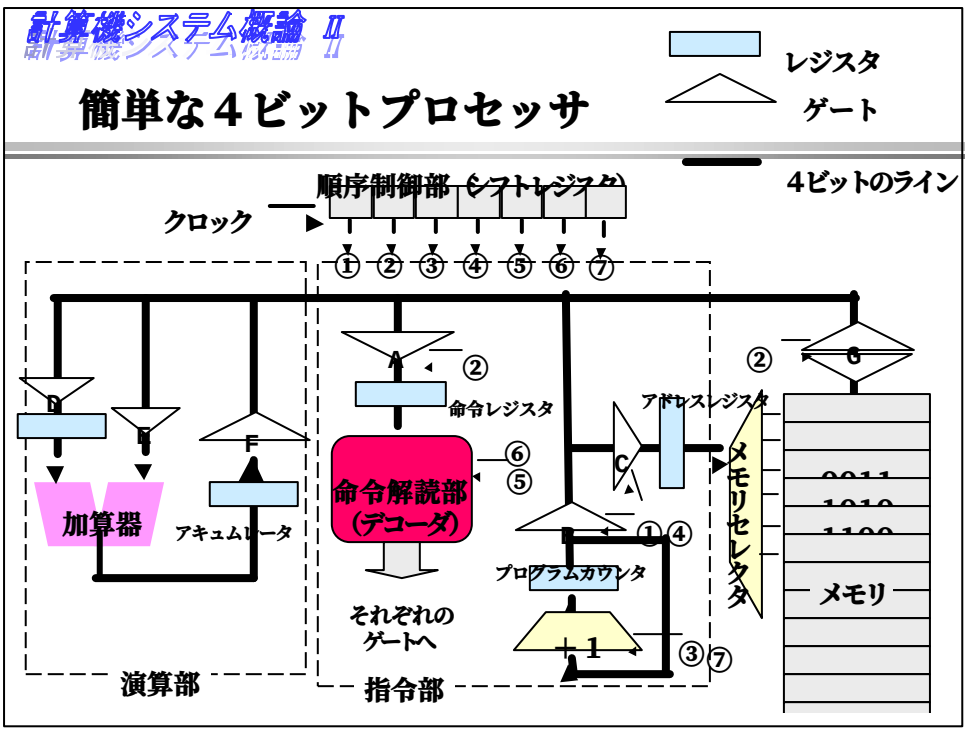
プログラムを実行するプログラムがつくれる⇒システム

コンピュータの基本的な構成

- 記憶（メモリ）、指令、演算



簡単な４ビットプロセッサ



説明

- 命令やデータのとおり道（実際は電線）を「バス」という。
  - データのとおり道はデータバス、
  - メモリにどのデータを読み出すかを伝えるバスをアドレスバス
- メモリの横にある「番人」（セレクタという）に接続されているのがアドレスバスで、操作するメモリを指定していま三角でしめされているのがゲート。信号の流れを制御する。
- このゲートを制御する信号は、現在の命令（０と１の組み合わせ）から、命令解読部（デコード）で作られる
- プロセッサの中にも一時的にデータを格納するメモリ（のようなもの）がある。レジスタと呼ばれる。そのいくつかはプログラムからは見えない（例えば、現在の命令を保持している命令レジスタや読み出すメモリの番地を保持しているアドレスレジスタなど）
- 実行するプログラムの番地を保持しているレジスタをプログラムカウンタという
- 演算の一時的な結果を保持するレジスタをアキュムレータと呼ぶことがある。実際のプロセッサではこのようなレジスタが複数ある。

計算機システム概論 II

命令コード（機械語）

- メモリ上にあるプログラムのそれぞれの命令は、動作とその対象からなる。
  - 動作を指定するのが、命令コード（オプコード）
  - 対象をオペランドという
  - 実際のマシンではオペランドのない命令もある
- このマシンでは、２ワードであらわす

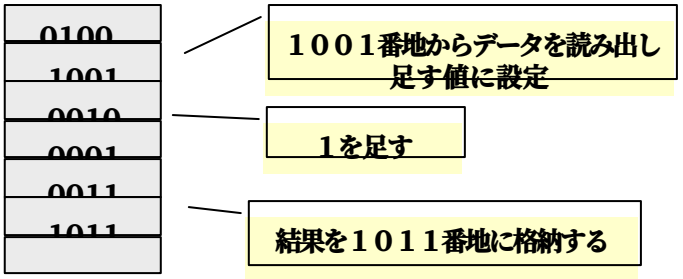
0001	X	Xを足される数に設定しなさい（LOADI）
0010	Y	設定されている足される数にYを足しなさい（ADDI）
0011	Z	Zのアドレスに結果を入れなさい（STORE）
0100	W	W番地の中身を、足される数に設定しなさい（LOAD）

どのように実行されるか

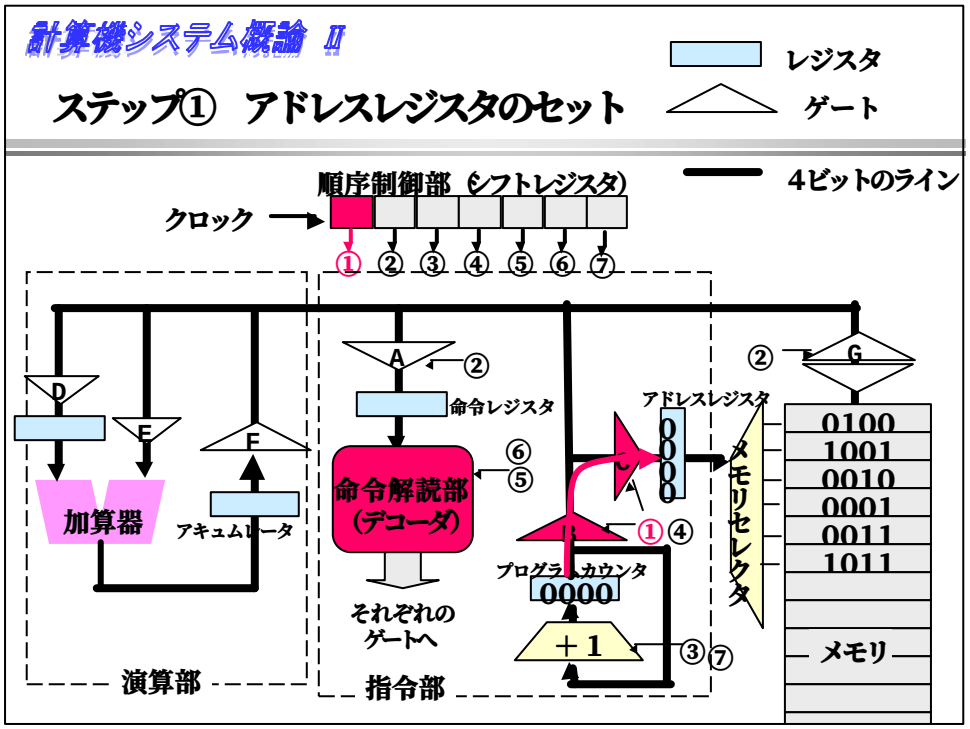
- クロック信号が入力されると、順序制御部から①から⑥までの信号が順番に送られる
  - ①　プログラムカウンタをアドレスレジスタに設定
  - ②　命令コードのフェッチ
  - ③　プログラムカウンタを１つあげる（オペランドを読む準備）
  - ④　プログラムカウンタをアドレスレジスタに設定
  - ⑤　命令コードの解読と実行１
  - ⑥　命令コードの解読と実行２
  - ⑦プログラムカウンタを１つあげる（次の命令を読む準備）
- これが、コンピュータの速度を決定する
- この単純なプロセッサでは、⑤⑥以外は同じパターンでゲートを開け閉めするが、⑤⑥だけは現在の命令から、デコード部で生成された信号でゲートを開け閉めする
  - このデコード部は組み合わせ回路である！

プログラム例

- ここで、番地１００１からデータを読み出し、１を加えて、番地１０１１に格納するというプログラムを考える

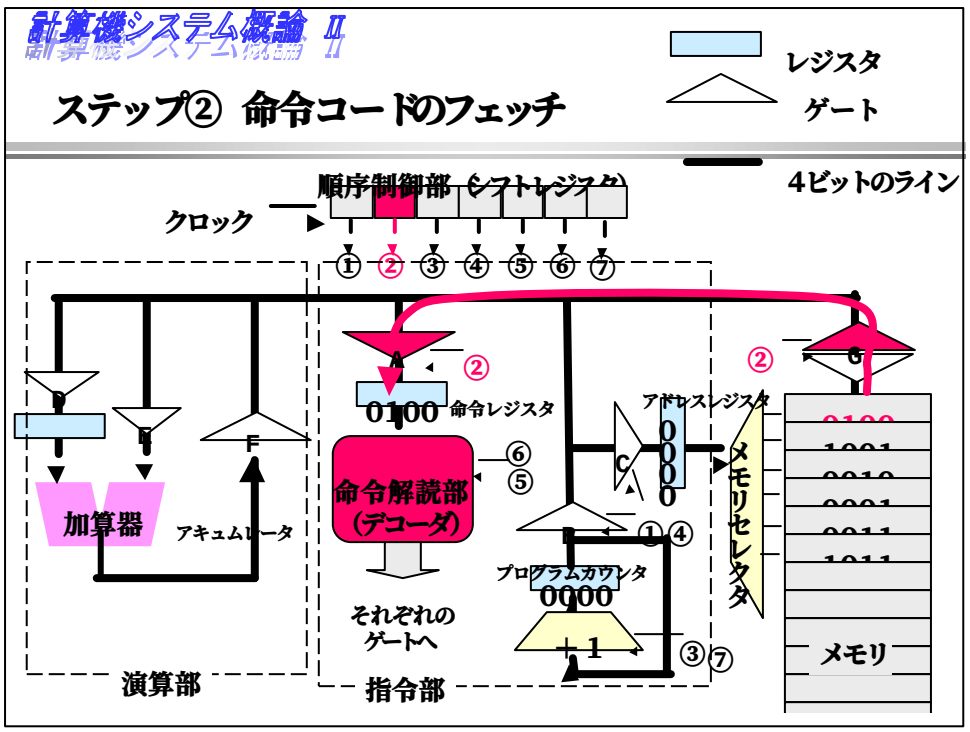


ステップ① アドレスレジスタのセット

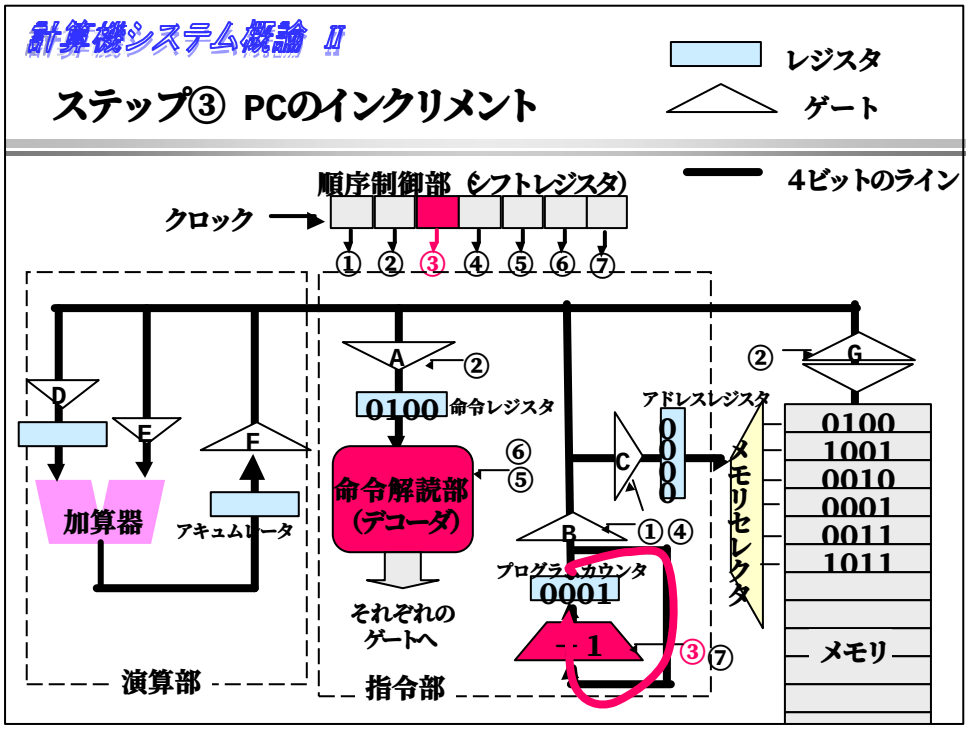


計算機システム概論 II

ステップ② 命令コードのフェッチ

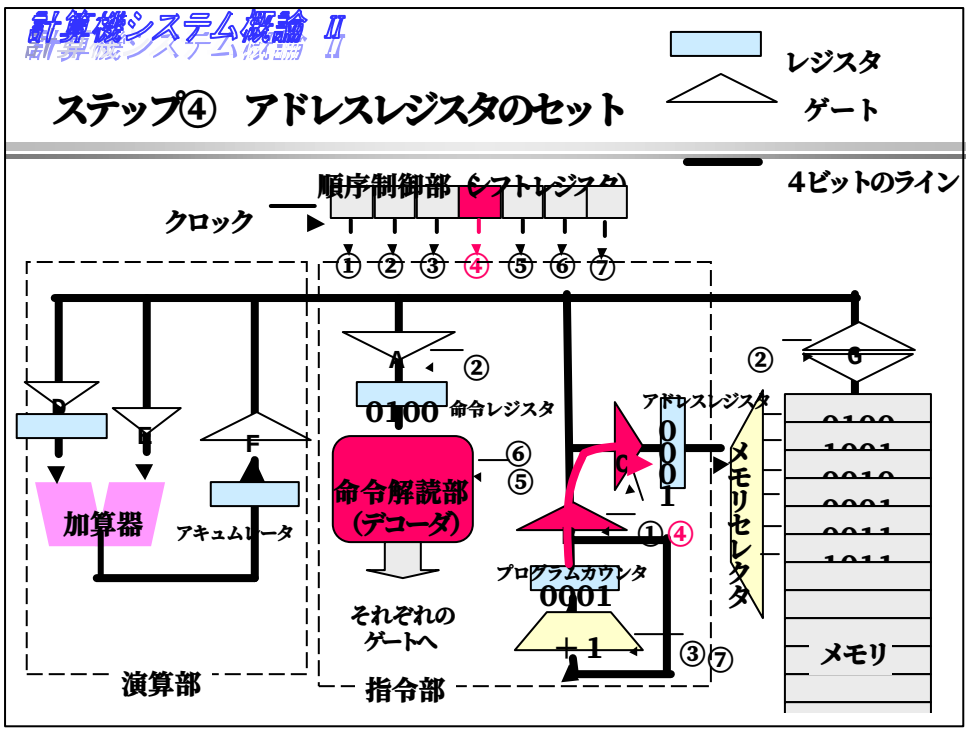


ステップ③ PCのインクリメント

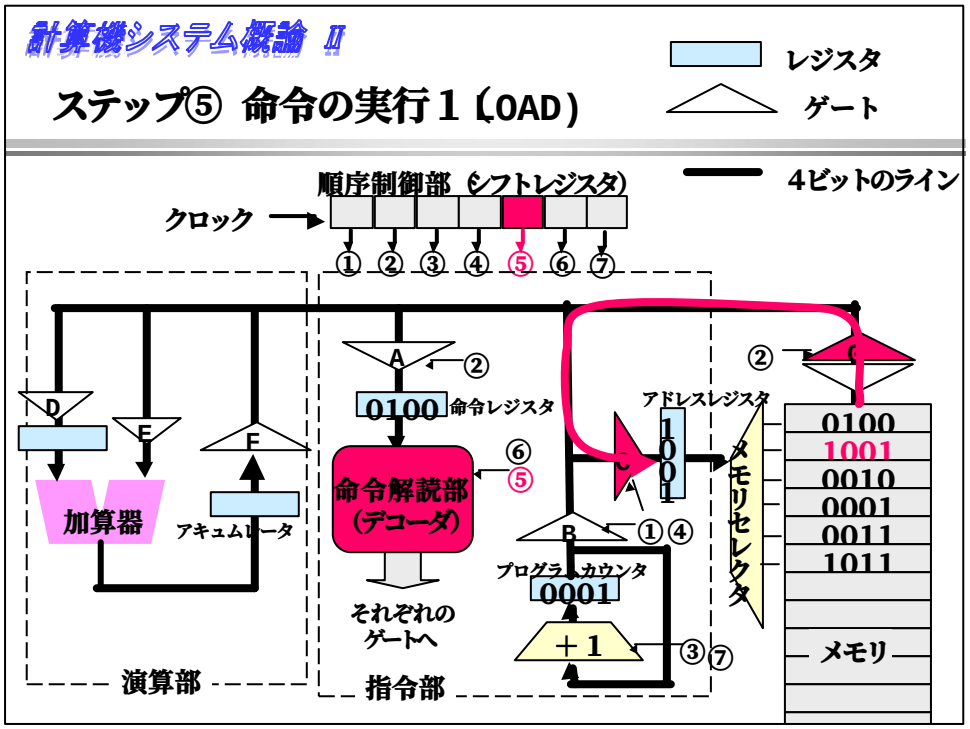


計算機システム概論 II

ステップ④ アドレスレジスタのセット

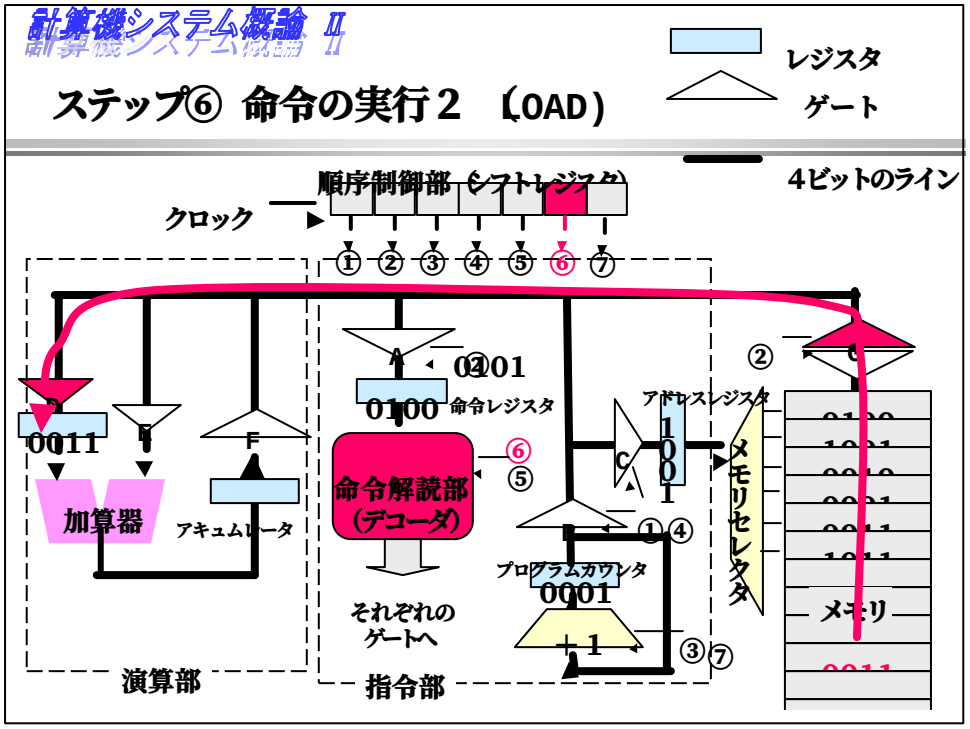


ステップ⑤ 命令の実行１（LOAD）

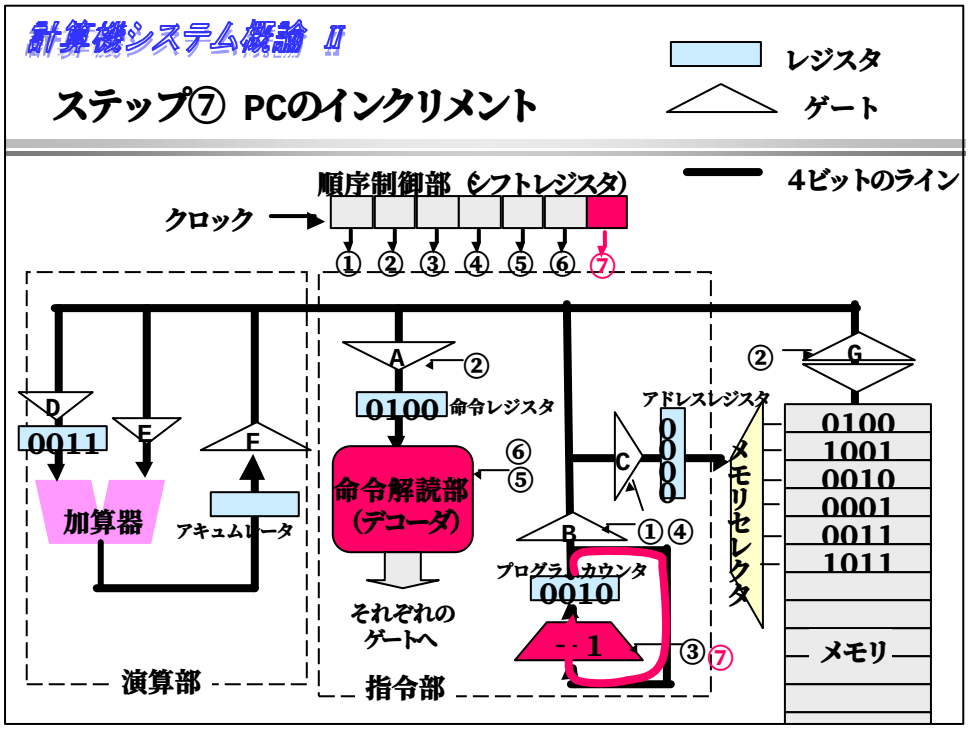


計算機システム概論 II

ステップ⑥ 命令の実行２ （LOAD）

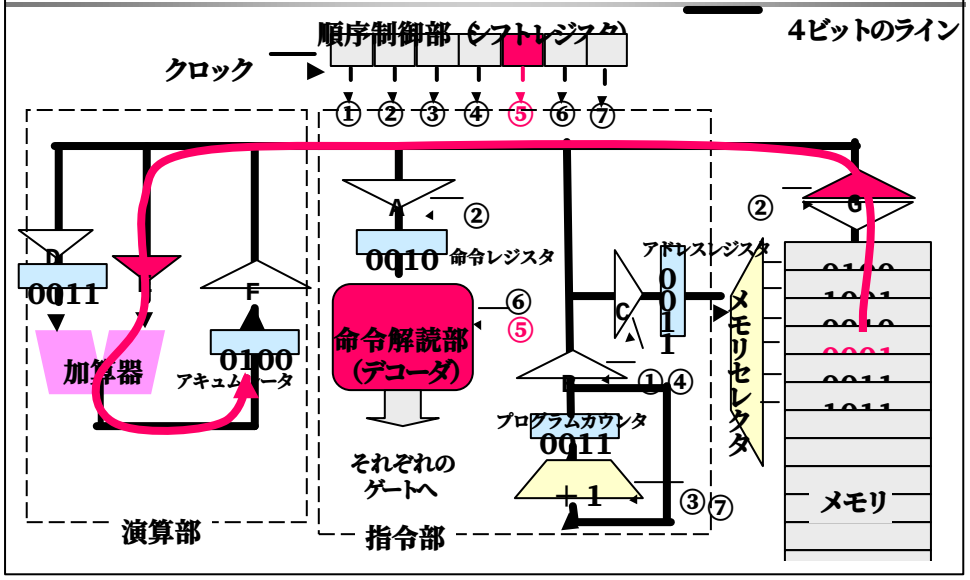


ステップ⑦ PCのインクリメント

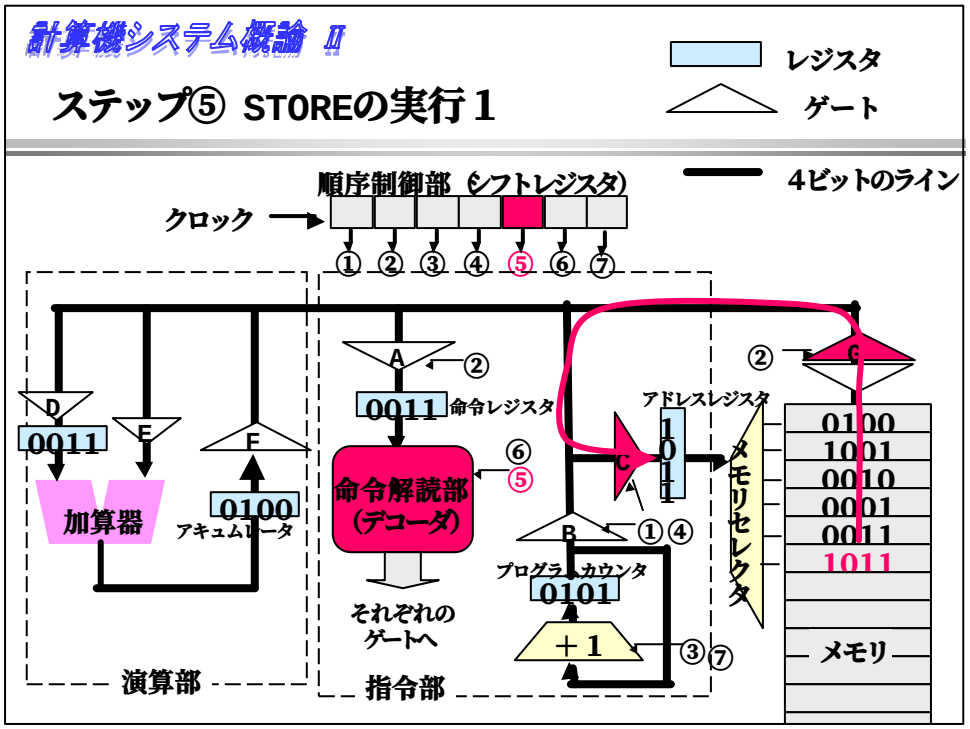


計算機システム概論 II

ステップ⑤ 加算の実行

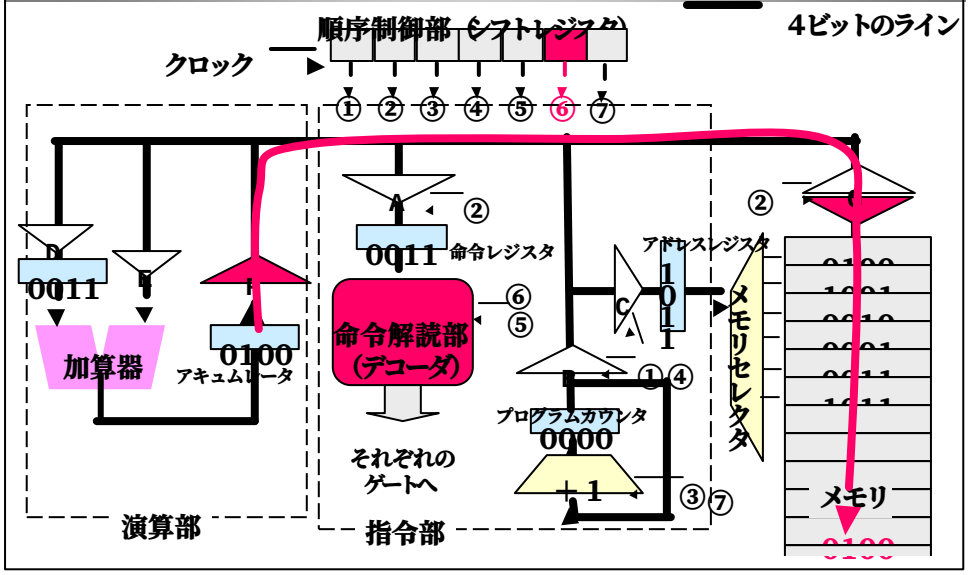


ステップ⑤ STOREの実行１



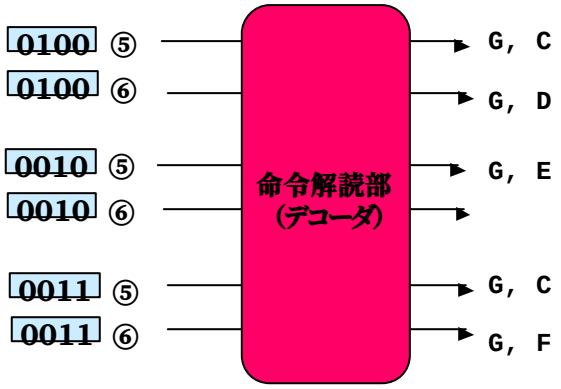
計算機システム概論 II

ステップ⑥ STOREの実行２



命令解読部（デコーダ）の働き

- 命令のデコーダは、基本的には組み合わせ回路



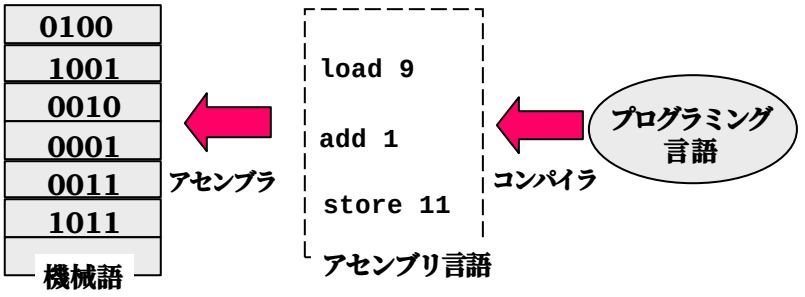
計算機システム概論 II

本当のマイクロプロセッサでは…

- 制御を変えるにはどうすればいい？(jump命令、条件分岐)
- 実際には、各ステップで同時に実行できるところがある。
  - たとえば、実行とプログラムカウンタのインクリメントを同時に実行
- プロセッサの実行フェーズ
  - 命令フェッチ　(IF : instruction fetch)
  - 命令デコード　(ID: instruction decode)
  - 命令実行　(EX: execute)
  - 結果の書き込み(WB: write back)
  - …
- レジスタがたくさんある。
  - 汎用レジスタ（整数とアドレス）
  - 浮動少数点レジスタ
- キャッシュメモリがある。
- オペレーティングシステムのいろいろな機能
  - 仮想記憶、割り込み、入出力

アセンブリ言語とコンパイラ

- マシン語をそのものでは０１のパターン、つまり数字ですので、これをつかってプログラミングするのは人間にとって非常に面倒な作業になる
- 基本的に、マシン語に１対１に対応するように記号を使って表記したのがアセンブリ言語
- オプコードを表す記号をニーモニックという
- コンパイラは、プログラミング言語をアセンブラ言語に翻訳する。



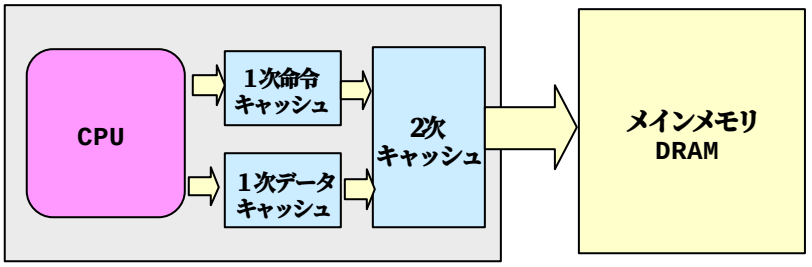
計算機システム概論 II

半導体プロセスとクロック速度

- 半導体プロセスの向上
  - ムーアの法則、半導体の集積度は１８ヶ月で２倍になる
  - 多くのトランジスタをつかうことができる
    ⇒いろいろな機能を盛り込む
  - Pentium4は、0.13μmプロセス
- クロック速度の向上
  - 明らか。（Pentium4は３GHz！）
  - 電圧を下げる　（５Vから３V)
  - 線幅が小さくなると回路を駆動する電流は小さくてすむが、電子の移動速度が遅くなる。

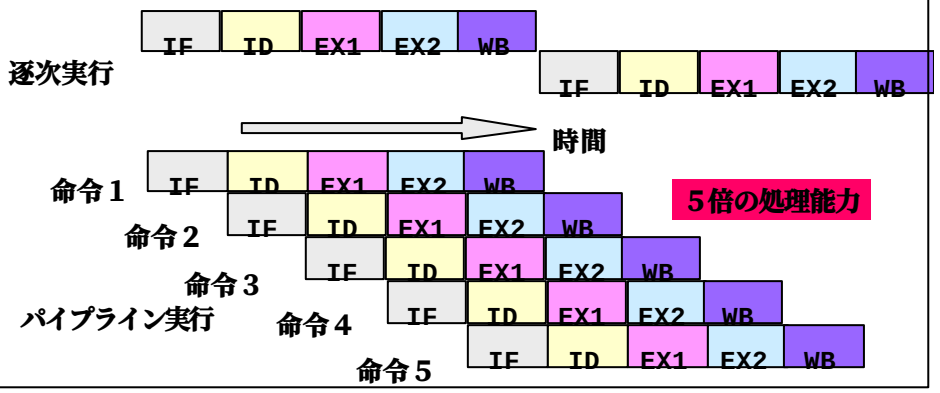
キャッシュメモリ

- 早い少量のメモリをCPUに近いところ（チップの中）におく。
  - キャッシュメモリ：SRAM(static RAM)、アクセス速度が速いが、少容量
  - メインメモリ：DRAM(dynamic RAM)、アクセスが遅いが大容量、安価
- 命令をおくための命令キャッシュとデータをおくためのデータキャッシュに分かれている
- １次キャッシュと２次キャッシュ、３次キャッシュも
- CPUのクロック速度が速くなっている現在、必須の技術



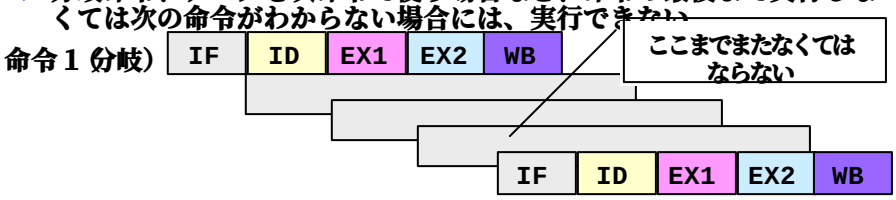
パイプラインアーキテクチャ

- 処理のフェーズをオーバーラップして、同時に複数の命令を時分割して実行する方式
- 同じ時刻ではそれぞれの命令は異なるフェーズを実行している。
- 細かくすればするほど、速度は向上する。
- 早いクロックのプロセッサでは各フェーズは細かくなる。

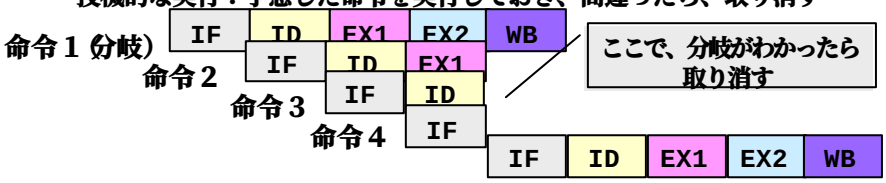


パイプラインアーキテクチャの問題点

- 分岐命令、データを次命令で使う場合など、命令の最後まで実行しなくては次の命令がわからない場合には、実行できない



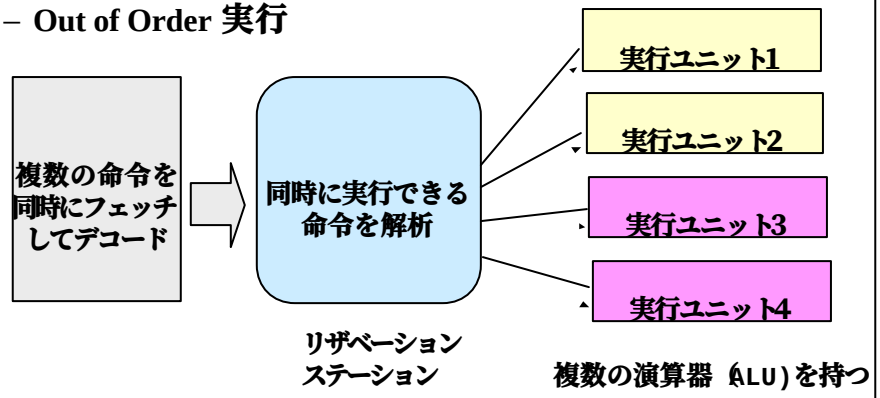
- 解決方法
  - 分岐予測：次に実行するかもしれない命令を予測する。
    - たとえば、繰り返しの分岐は実行されることが多い。
  - 投機的な実行：予想した命令を実行しておき、間違ったら、取り消す



計算機システム概論 II

スーパースカラ

- 複数の命令をフェッチして、並列に実行できる命令を見つけて、複数の実行ユニットを使って、複数の命令を同時に実行する。
  - Out of Order 実行



スーパースカラの問題点

- （CPUが複雑になる）
- パイプラインと同じく、分岐、データ依存があると実行できる命令数が少なくなる。
- 解決方法
  - リネーミングレジスタ：レジスタに書き込みがある場合に、別のレジスタに書いておき、後でつじつまを合わせる。

```
mov m1,r1     ; m1 <- r1
mov r1,m2     ; r1 <- m2
add r3,r1,r2  ; r3 <- r1+r2
add r5,r1,r4  ; r5 <- r1+r4
add r1,r2,r3  ; r1 <- r2+r3
mov m2,r6     ; m2 <- r6
```

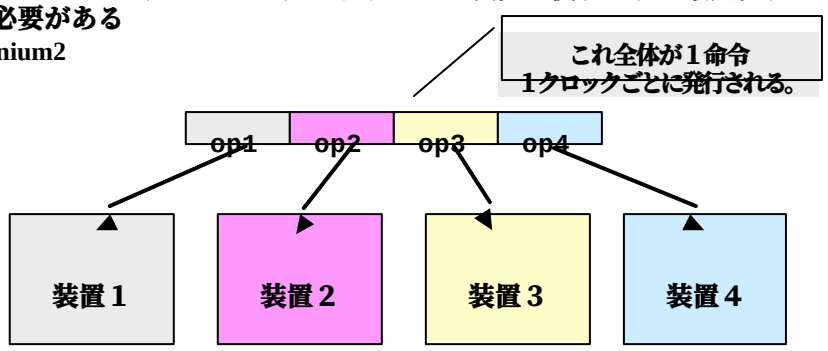
r1がm1に書き込まれるまで、r1にかきこめない。

r1を別のレジスタに書いておきそれをつかう。

計算機システム概論 II

VLIWアーキテクチャ

- VLIW (Very Long Instruction Word)アーキテクチャ
- 複数の演算装置を持ち、それぞれに対する命令コードを持つような、長い命令を使ったアーキテクチャのこと。
- ソフトウエア（コンパイラ）で、すべての装置を使うように最適化する必要がある
- Itanium2



特別な用途の命令

- グラフィック処理など、特別なアプリケーションに有効な命令を作る
  - たとえば、３ｘ３の行列演算、座標変換などに用いる
  - MMX (Multi-Media eXtension)
    ⇒ SSE (Streaming SIMD Extensions)
  - 3DNow! (AMD)
  - AltiVec (IBM, PowerPC)

まとめ

- CPUの基本的な仕組み
- CPUの高速化の技法
  - パイプライン
  - スーパースカラ
  - VLIW …
- 次回は、並列処理とグリッド