はじめに

- 前回までは、命令セットについて概観してきました。
  - マシン命令とはどういうものか、コンピュタハードウェアで何ができるのか、について学びました。
  - 命令セットを通して、コンピュータハードウェアの機能がどういうものか、を理解するのが目的でした。
- 今回は、マシン命令をハードウェアでどのように処理するのか、その仕組み（機構）について講義します。

- まず、最初の話題についての教科書の対応範囲は、以下の通りです。
  - 2.2.5項（p.55～57）

命令実行サイクル

- 1個のマシン命令を実行する過程を命令実行サイクルといいます。
  - 大まかに分けると、以下のステージを順に進めます。
    1. 命令フェッチ（IF: Instruction Fetch）
    2. 命令解読（D: instruction Decode）
    3. オペランドフェッチ（OF: Operand Fetch）
    4. 演算実行（EX: EXecution）
    5. 結果格納（WB: Write Back）
    6. 次命令アドレスの決定
  - 電源が入っている間中、プロセッサはこの命令実行サイクルを繰り返します。

命令フェッチ

- コンピュータ(プロセッサ)は単に計算だけを行うわけではありません。
  - 色々な計算(加算/減算/乗算…など)ができるということは、裏を返せば、そのどれを行うのかを指示しなければならない、ということ。
  - その指示を行うのがマシン命令/命令語です。
  - ですから、いきなり計算を始めるのではなく、まずは、命令語をメモリから読み出さなければなりません。この処理を命令フェッチと呼んでいます。
    - 本来なら計算だけをさせたいのですが、こういう余分な処理も必要になります。
    - 当然、計算に要する時間だけでなく、このような処理を行う時間もかかる、ということです。
      - 汎用性(なんでもできる)というのは、こういう処理の必要性と引き換えに得られるものなのです。

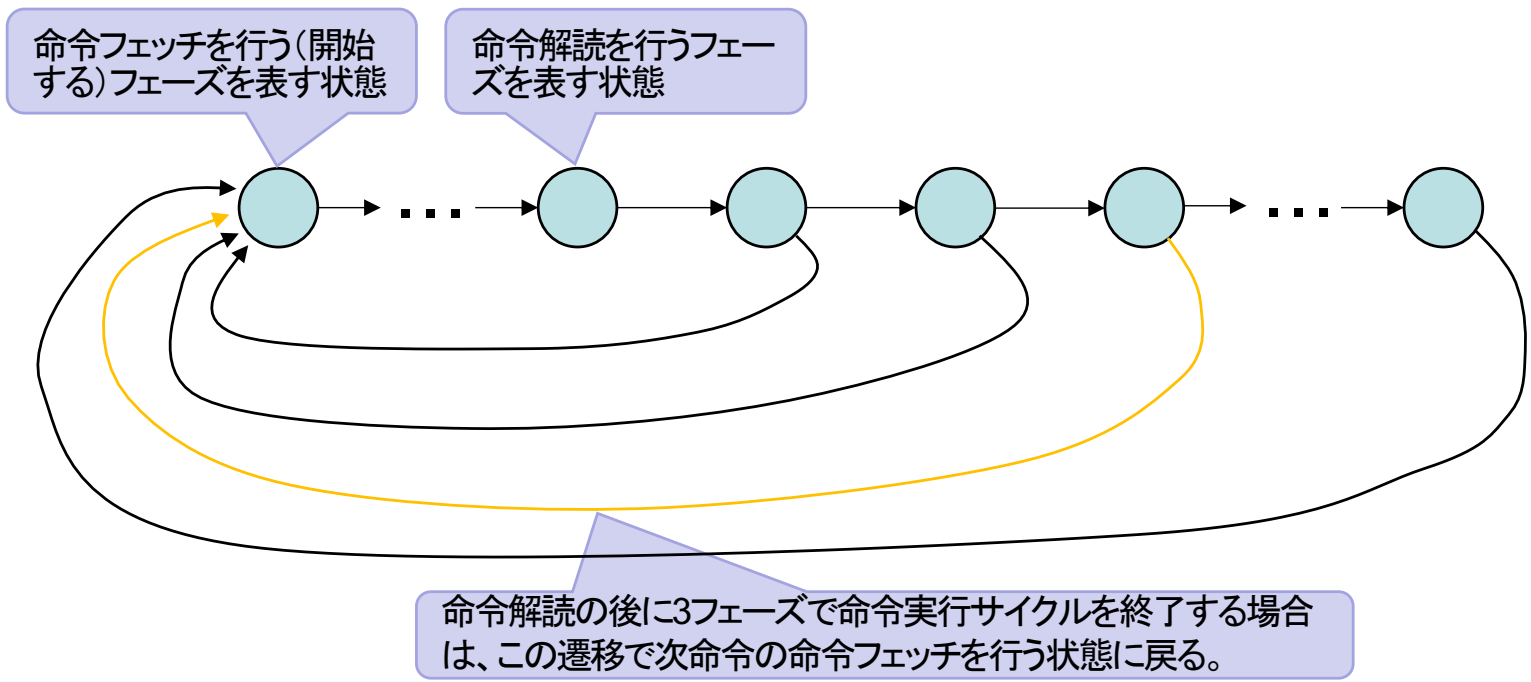
試験の解答などでは「読み出す」ではなく「呼び出す」などという人がいますが、全く意味が異なります。技術者は正確に言葉を使わなければなりません(話が伝わらない人として相手にされなくなります)。「読み出す」だけでなく、他の場合でも、意識しなくてもできるようになるまでは、特に意識して注意してください。

- 命令フェッチとして行う処理の内容は以下の通りです。
  - 次に実行するべき命令語が格納されているアドレスは、プログラムカウンタ(PC)に格納しています。
  - したがって、PCの値(アドレス)をメモリに送り、その番地の命令語を読み出します。
  - メモリから読み出した命令語を、プロセッサ内の命令レジスタ(IR: Instruction register)に格納する。
    - 命令が可変長の命令セットの場合は、これから読み出す命令語の長さが何バイトかわかりません。したがって、まず命令語の1バイト目を読み出し、その中に書かれている命令語長に基づいて残りの部分を読み出す、などの複雑な処理を行う必要があります。
      - メモリに2回以上アクセスすることになり、それだけ時間もかかります。

命令解読

- メモリから読み出した命令語を解読します。この時点で初めて、命令語の内容を理解します。
  - » いえ、機械ですから本当に理解などしているわけではありませんが。
- IR内の命令語を解読して以下のことを行います。
  - OPコードなどから、この命令を処理するのにあと何サイクル（各サイクルをフェーズと呼ぶことにします）かかるかを求め、フェーズ信号生成器をセット・起動します。

- フェーズ信号生成器とは？
  - 命令実行サイクルにおける各フェーズ（サイクル）を識別する信号を出力する順序回路です。
  - （注） ここでは「フェーズ信号生成器」と呼ぶことにしますが、特に共通してそう呼ぶわけではありません。
  - 例えば、以下のような状態遷移図にしたがって動作する単純な順序回路として実装できます。



- フェーズ信号生成器の出力とIR内のOPコードなどから、各部を制御するための制御信号を生成する。
  - プロセッサ内の各ブロックは、この制御信号に基づいて動作します。
  - 各ブロックの動作は、フェーズごとに異なります。
  - 例えば、IR内の命令が減算命令だったとしても、その命令実行サイクル中の全てのフェーズで、演算器が減算を行うわけではありません。処理する命令（の種類）とどのフェーズかの組み合わせで、各ブロックの動作が決まります。
    - 典型的には、IRの内容とフェーズ信号生成器の状態信号を入力とする組み合わせ回路にとって、そのフェーズにおける制御信号を作り出します。

オペランドフェッチ

- アドレッシングモードに従って、演算に使用するデータの値を用意します。
  - ソースオペランドがレジスタの場合は、レジスタから値を読み出します。
  - ソースオペランドが即値の場合は、IR内の即値の値に符号拡張を施して、データサイズを整えます。
  - ソースオペランドがメモリの場合は、アドレス計算を行って実効アドレスを求め、メモリからデータを読み出します。
    - 複雑なアドレッシングモードの場合は、実効アドレスを求める過程で加算やメモリアクセスが必要になるため、複数のフェーズに分けて処理を行います。

演算実行

- 演算器で演算を行います。
- 演算器関係の名称について
  - 演算ユニット（Functional Unit）
  - 加算器（Adder）、乗算器（Multiplier）、除算器（Divider）
  - 算術論理演算ユニット（ALU: Arithmetic Logic Unit）
    - 本来は、固定小数点数に対する加減算・論理演算を行う組み合わせ回路を指す名称です。
    - ただし、固定小数点数に限らず、広い意味で「演算器」を指す名称として使用する場合があります。

結果格納

- 演算器の出力（演算結果）をデスティネーションオペランドで指定された場所に格納します。
  - デスティネーションオペランドがレジスタの場合は、レジスタに演算結果を書き込みます。
  - デスティネーションオペランドがメモリの場合は、アドレス計算を行って実効アドレスを求め、メモリに演算結果を書き込みます。

PC更新

- 次命令アドレスを求め、PCに格納します。
  - 通常の命令の場合は、現命令アドレス（現在のPCの値）に現命令語のサイズを加算したものが、次命令アドレス（更新後のPCの値）になります。
  - 分岐命令で分岐する場合は、アドレッシングモードにしたがって分岐先の命令アドレスを計算し、PCに格納します。
  - 高速化のため、PC更新用に専用の加算器を備えて、次命令アドレスの計算を他のステージと同時に行う場合があります（現在ではむしろこちらが主流）。

マシン命令の処理例

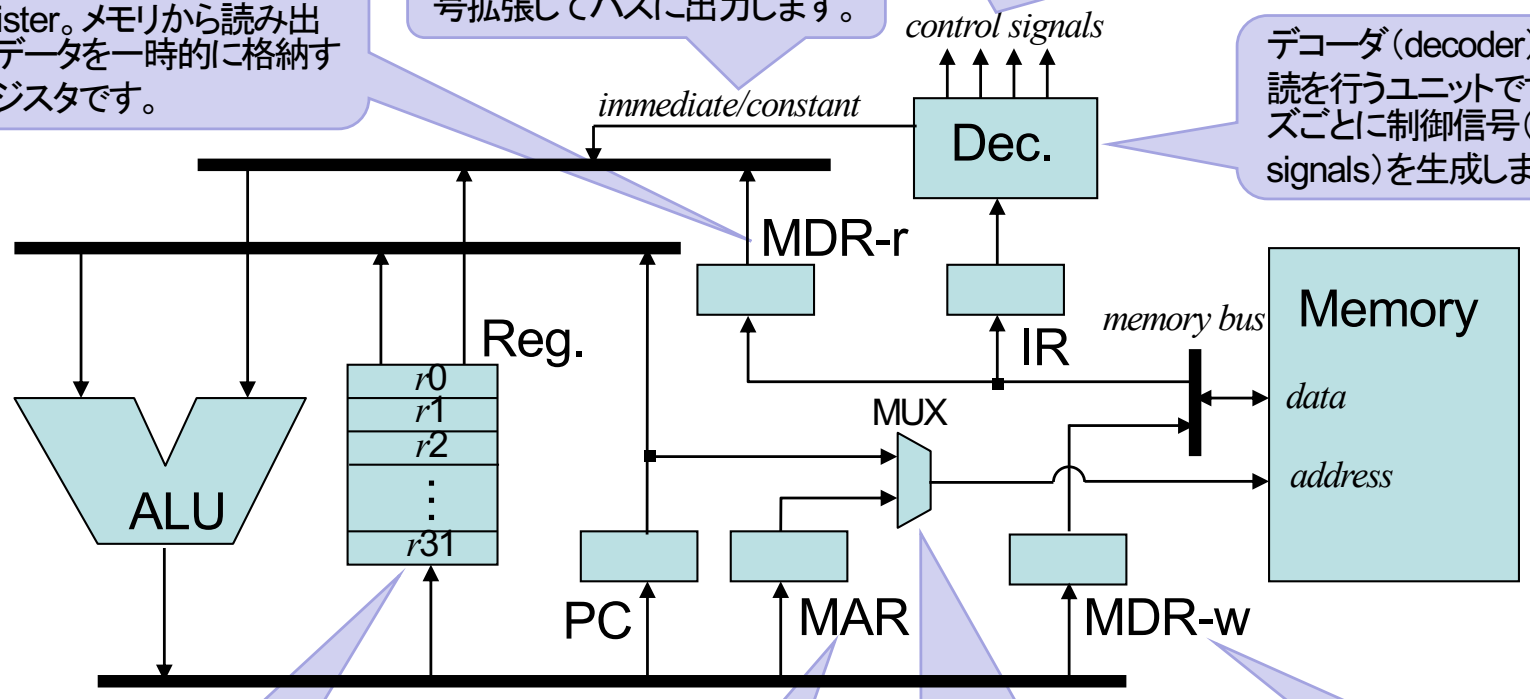
- では、次ページの単純な3バス構成のコンピュータハードウェアでどのようにマシン命令を処理するか、具体的な例を用いて説明していきます。
  - メモリ（主記憶）とメモリバス以外は、全てプロセッサ内に含まれます。
  - できれば論理回路をイメージして、動作を追いかけてください。

論理回路でこういう風に作れるなぁ、とイメージしてください。実際に細かく設計しなくても、組み合わせ回路なら真理値表、順序回路なら状態遷移図のイメージがつかめれば十分です。

プロセッサ内の各部へ送る制御信号（control signals）。煩雑になるので配線は省略していますが、各部の制御に必要な信号は全てデコーダで生成しますので、デコーダが司令塔に当たります。他のユニットはデコーダからの制御信号にしたがって動作するだけです。例えば、ALUでどんな種類の演算を行うのかを指定する信号や、MUXで入力のどれを選択して出力するのかを指示する制御信号、レジスタ（PCやIRも含む）ごとにその内容を更新するか否かを指示する制御信号など、すべての制御信号を含みます。

読み出し用のMemory Data Register。メモリから読み出したデータを一時的に格納するレジスタです。

オペランドの即値や定数を符号拡張してバスに出力します。



デコーダ（decoder）。命令解読を行うユニットです。フェーズごとに制御信号（control signals）を生成します。

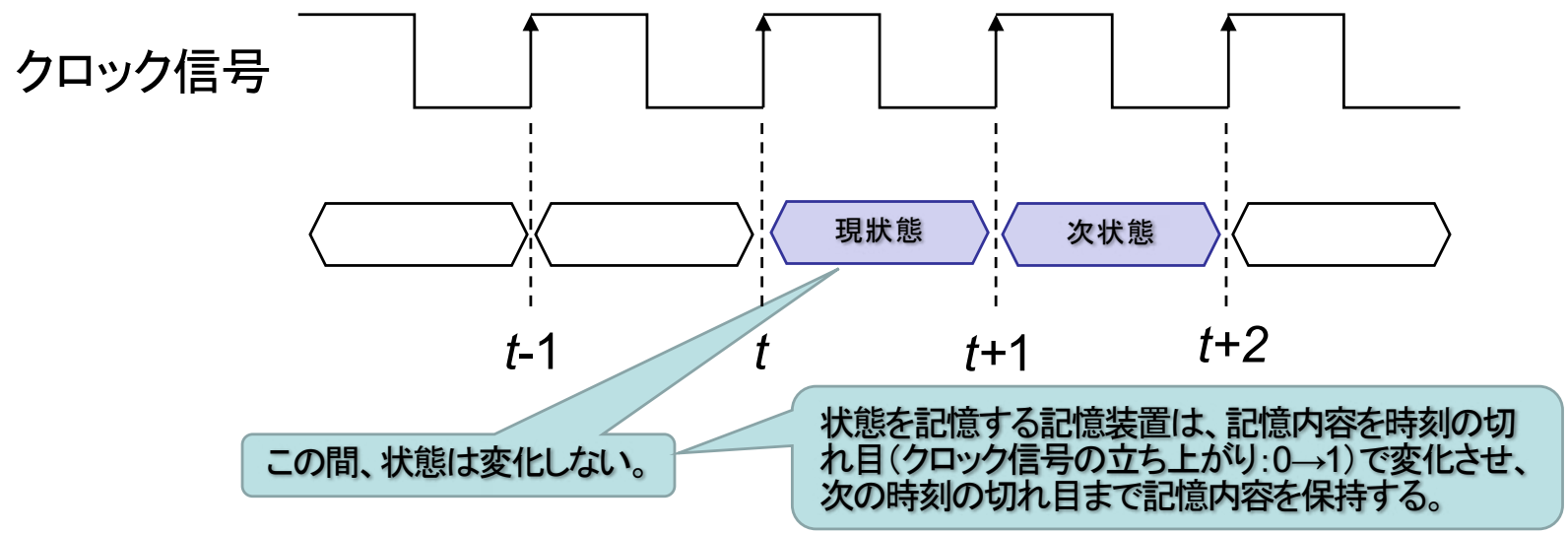
汎用レジスタ。0～31番の計32本の固定小数点レジスタがあるものとします。

Memory Address Register。メモリにアクセスするためのアドレスを一時的に格納するレジスタです。

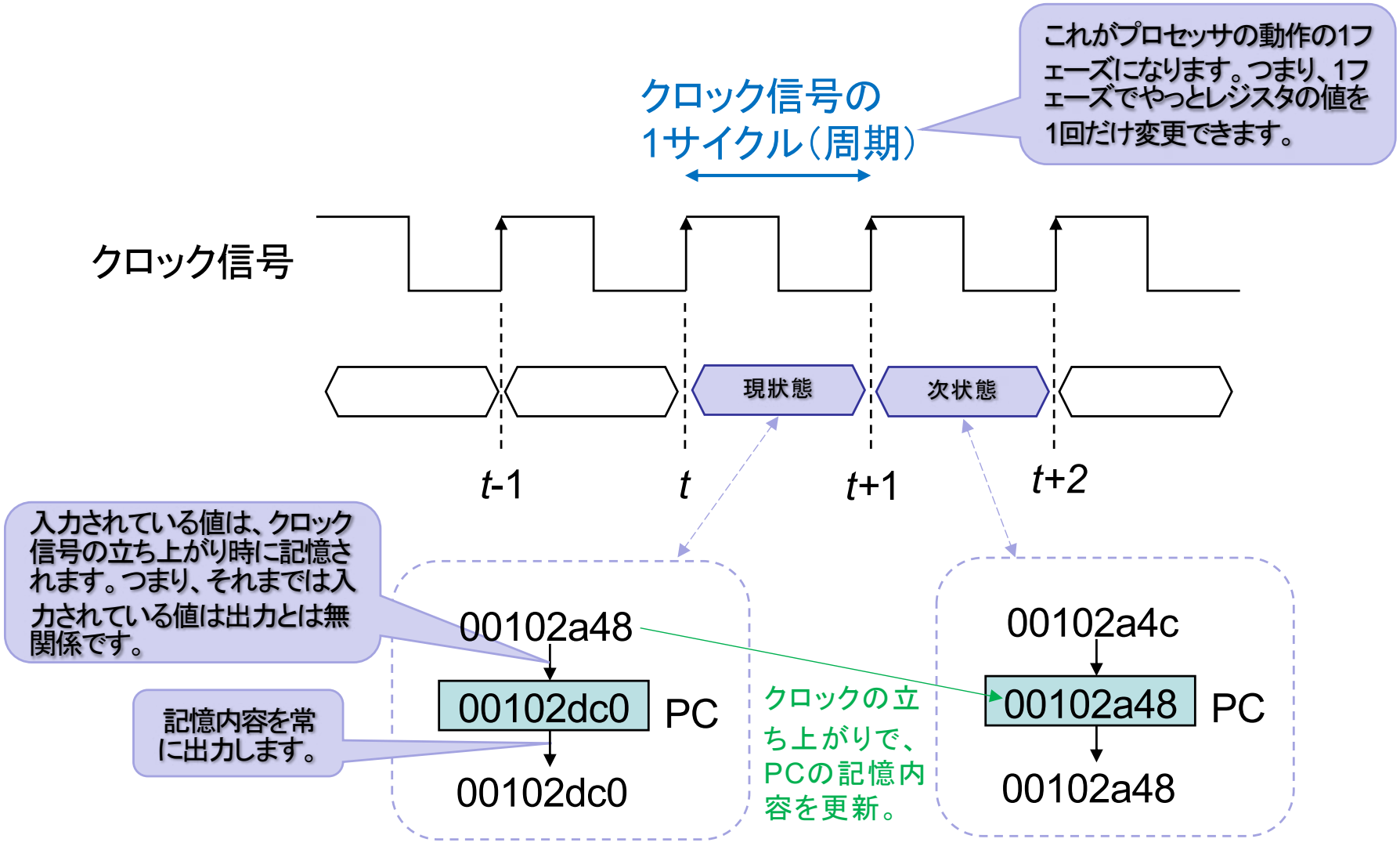
2-to-1のマルチプレクサ（セレクタ）。2つの入力のいずれか一方を選択して出力します。どちらを選択するかを指示する信号（control signals）はデコーダから送られてきます。

書き込み用のMemory Data Register。メモリに書き込むデータを一時的に格納するレジスタです。

- まず、同期式順序回路についておさらい(「論理設計」の授業で学びました)しておきます。
  - プロセッサ自体が、大規模な同期式順序回路です。
    - しかし、プロセッサ全体についての状態遷移図を書くような複雑なことをする必要はありません。
    - 状態遷移図を書いて設計しなければならないような機能ブロックがあるとすれば、それは先に説明したフェーズ信号生成器ぐらいです。
- とにかく、基本として、状態が変化するタイミングは、クロック信号の立ち上がり(または立ち下がりに統一)です。

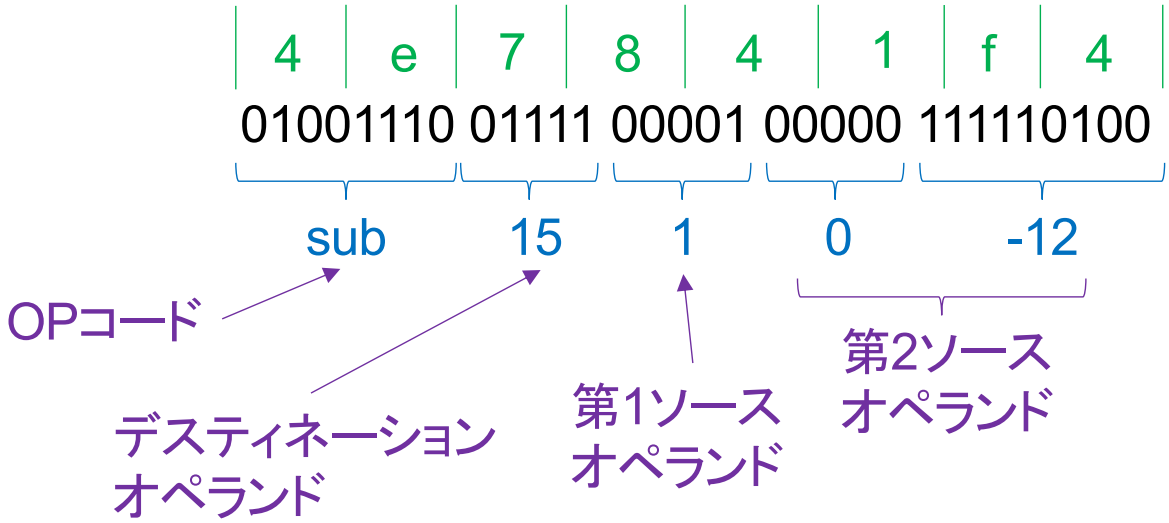


- レジスタの動作を考えると、



- では、以下の命令の実行サイクルを追いかけます。
  - 32ビットのプロセッサを想定。

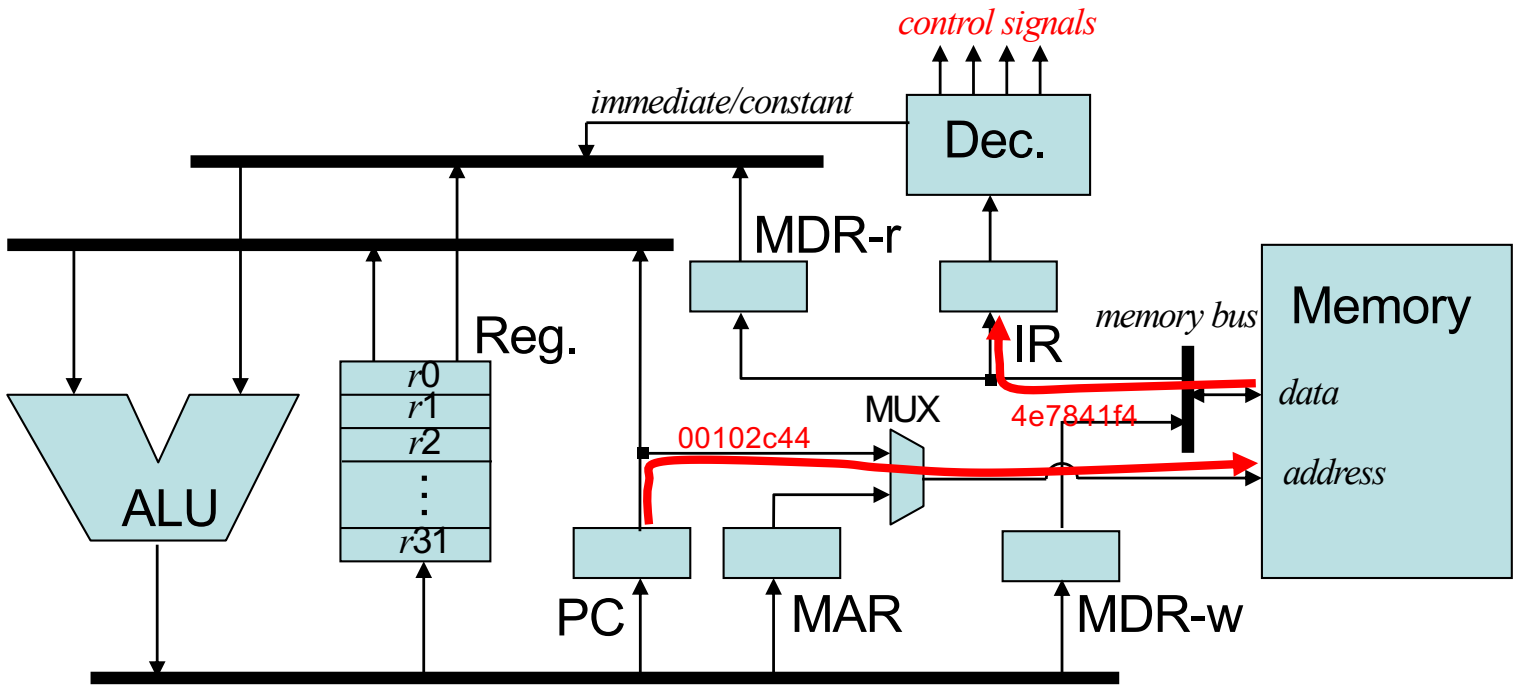
  - 命令アドレス: 16進表記で00102c44番地
  - 命令語: 16進表記で 4e7841f4



  - アセンブリ表記: sub r1, -12(r0), r15
  - 機能(動作): r15 ← r1 − Mem(r0 - 12)
    - レジスタr1の内容からメモリに格納された値を引いて、その結果をレジスタr15に格納する。引くメモリの値は、レジスタr0の内容をベースアドレスとして、それから12を引いた番地に格納されている32ビットのデータである。

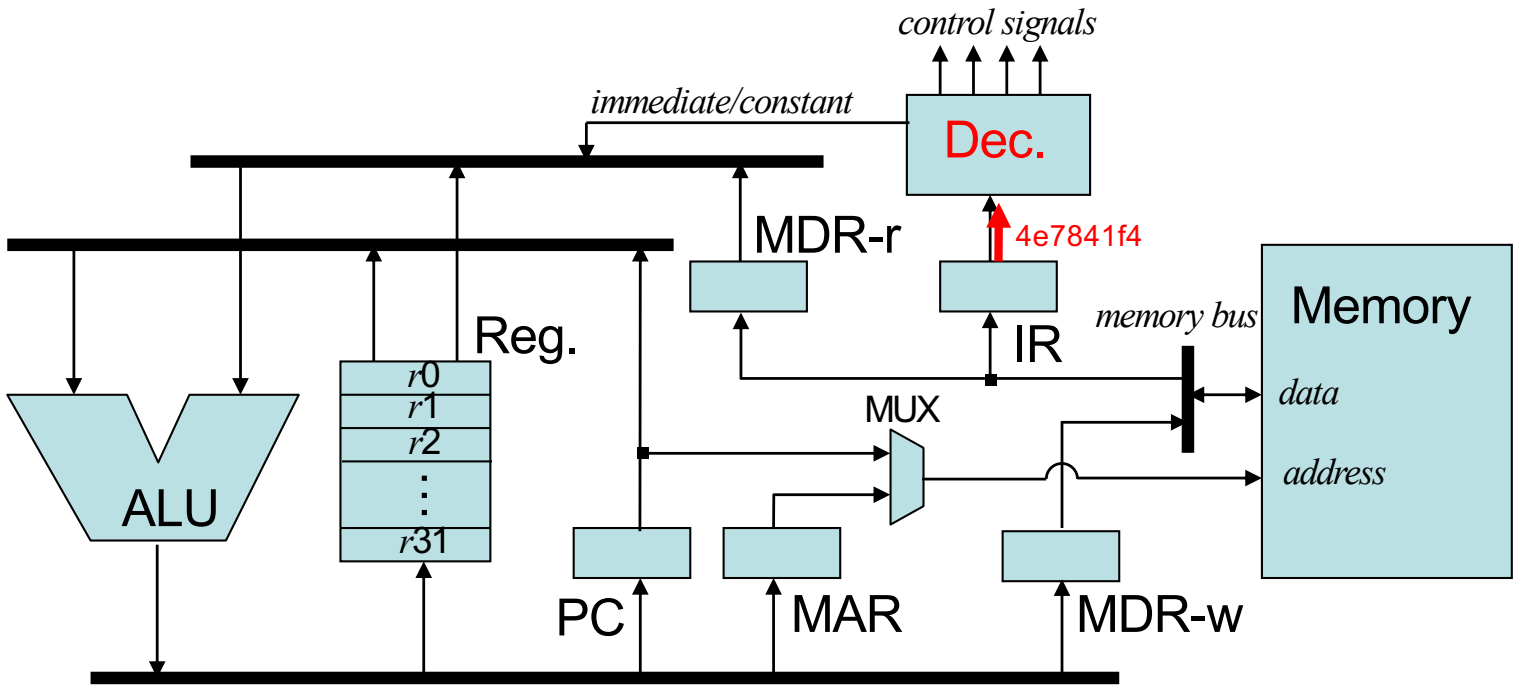
フェーズ0）

- プログラムカウンタPCに格納されているアドレスをもとに、メモリの00102c44番地から命令語4e7841f4を読み出して、命令レジスタIRに格納。
- 命令フェッチのステージの処理。



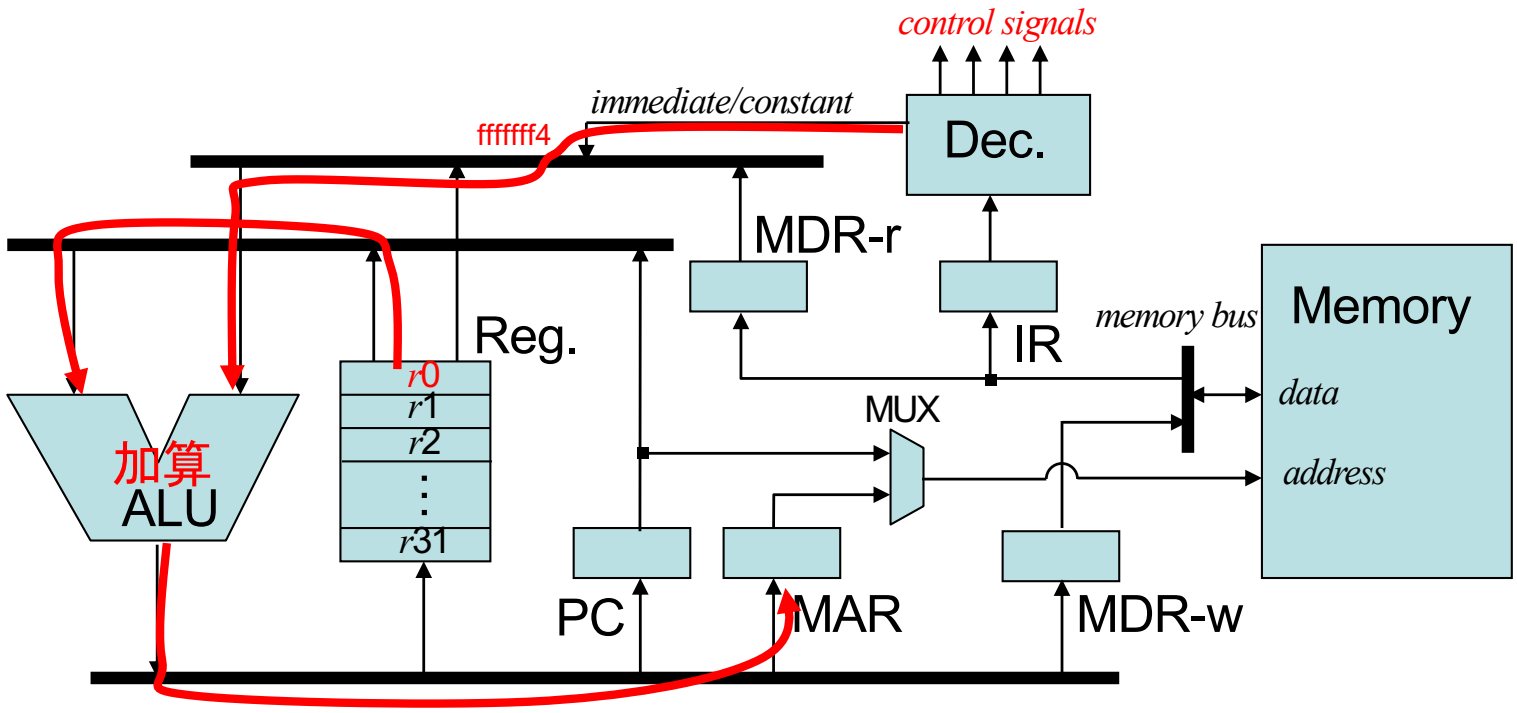
フェーズ1）

- 命令レジスタIRに格納した命令語4e7841f4をデコーダで解読。
- 命令デコードのステージの処理。



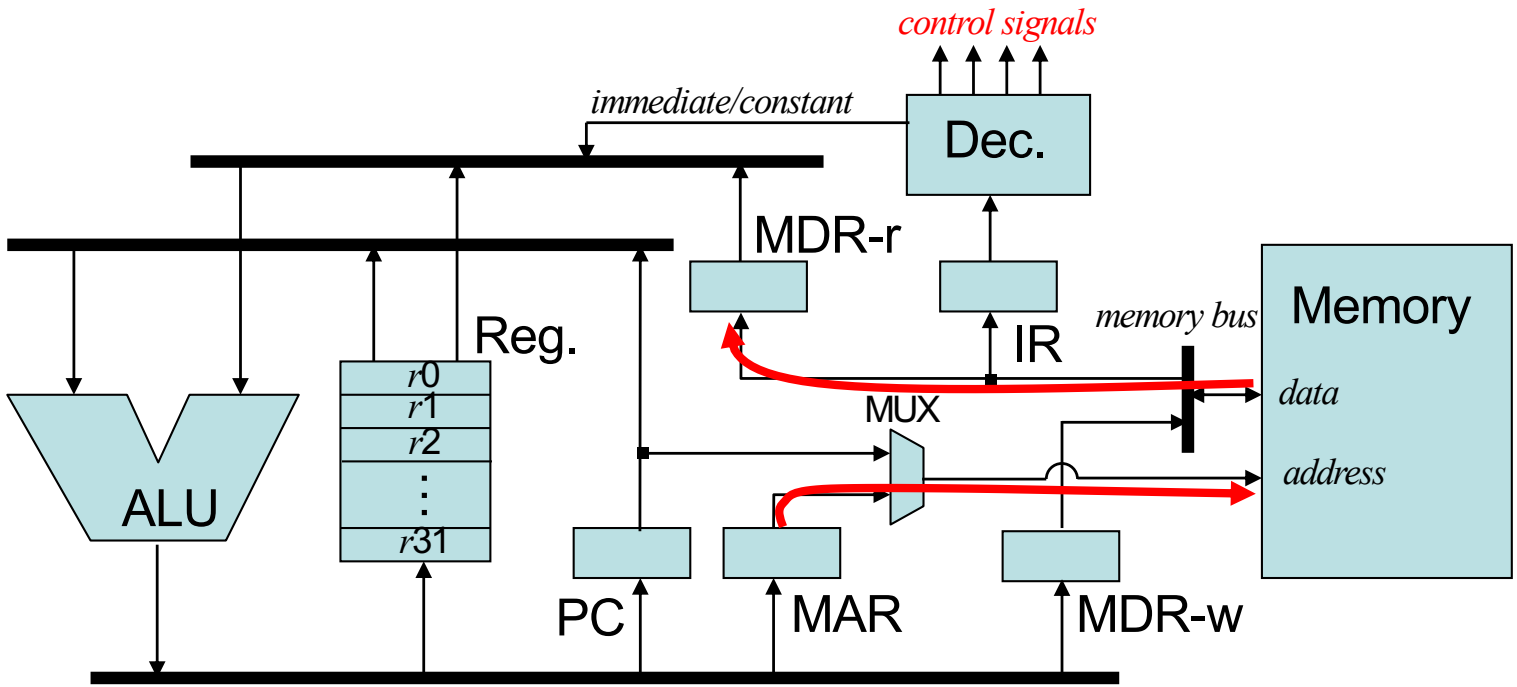
フェーズ2）

- レジスタr0の内容と−12(fffffff4)とをALUにて加算し、その結果をMARに格納。
- オペランドフェッチのステージの処理（アドレス計算）。



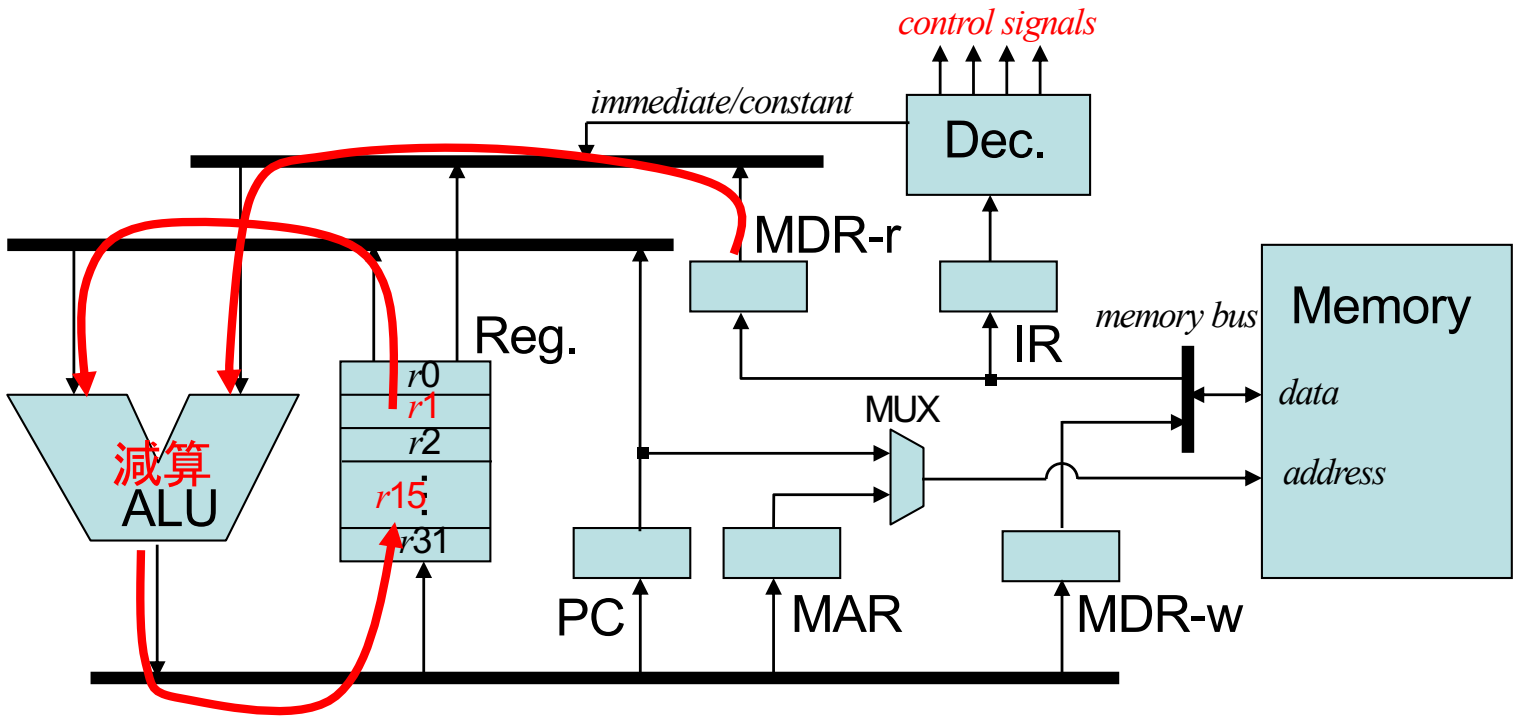
フェーズ3）

- MARに格納されているアドレスをもとに、演算に使用する4バイトのデータをメモリから読み出して、MDR-rに格納。
- オペランドフェッチのステージの処理（メモリアクセス）。



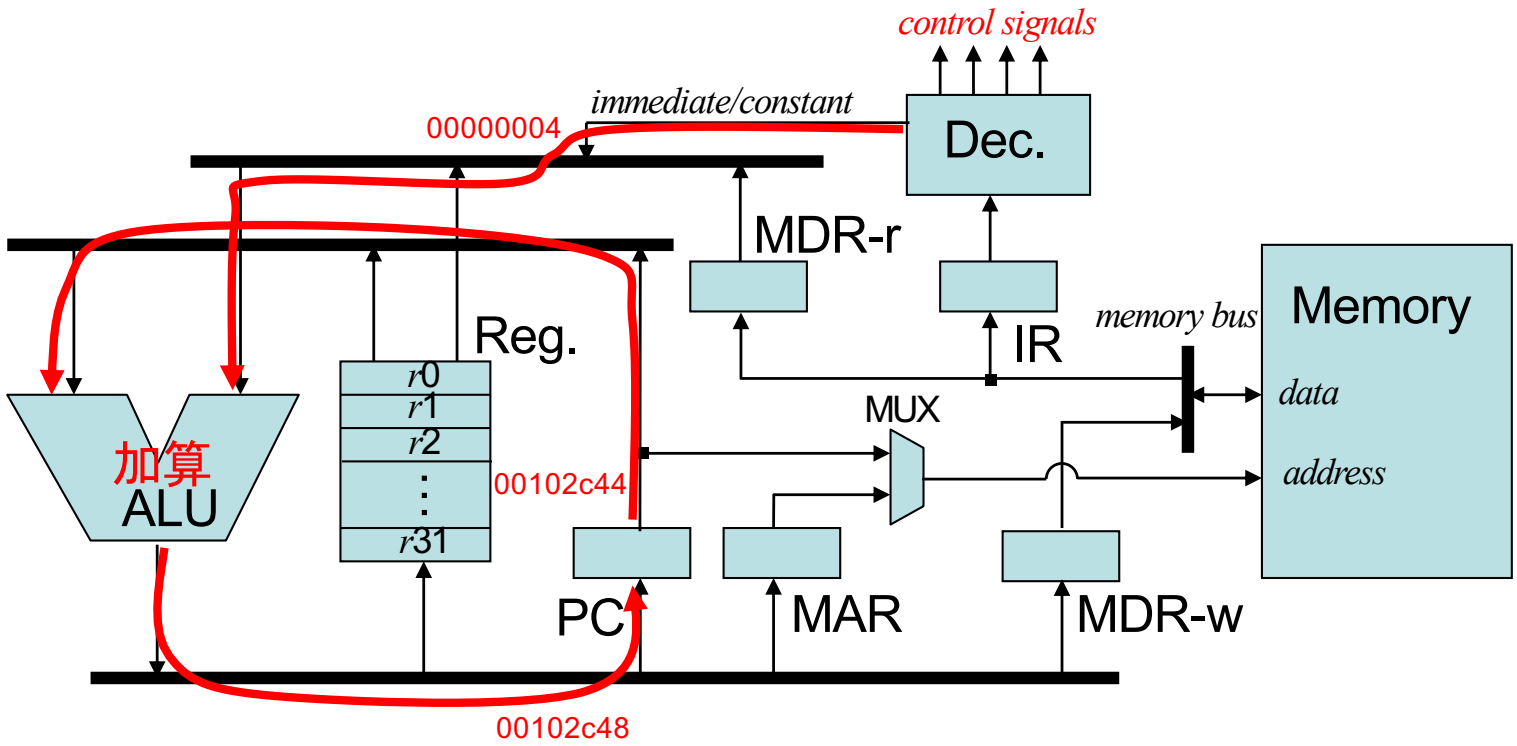
フェーズ4）

- レジスタr1に格納されている値とMDR-rに格納されている値とをALUで減算し、その結果をレジスタr15に書き込む。
- 演算実行および結果格納のステージの処理。



フェーズ5）

- プログラムカウンタPCに格納されているアドレス00102c44に4を加えて、次命令のアドレスを求める。
  - これでこの減算命令の処理は終了。次サイクルはフェーズ0に戻り、00102c48番地の命令の実行サイクルが始まる。
- PC更新のステージの処理。



- 1個のマシン命令を処理するのには複数サイクルの時間がかかります。
  - この例の減算命令の場合は、フェーズ0からフェーズ5まで、合計6サイクルをかけて1命令を処理しました。
- 当然、命令機能が複雑であれば、その命令の処理にかかるサイクル数も多くなります。
  - つまり、プログラムの実行時間は、実行するマシン命令の数（これを「動的な実行命令数」と言います）だけによって決まるわけではありません。
  - 複雑なマシン命令を使用すれば動的な実行命令数を減らすことができますが、その複雑な命令を処理するのに多くの時間がかかります。