3. コンピュータ(計算機)

この章では、1 章で説明したように表現されている情報に対して、2 章で説明したような情報の処理方法をどのようにコンピュータ内部で行っているかを説明する。

一般的なコンピュータ(計算機)の構成・機能、データ処理の流れ関する基礎知識を整理する。まず、コンピュータの種類を整理する。

コンピュータの種類と特徴

種類	特徴
スーパーコンピュータ (Super Computer)	大規模な科学計算技術計算用に設計。 最も高速・高性能なコンピュータ。
汎用コンピュータ (General Purpose Computer)	事務処理から技術計算までの多目的に利用できるよう設計された大型コンピュータ。高い耐故障性や故障時の代替処理の迅速化を考えて設計されている。**メインフレーム**ともいう。
ワークステーション (WS; Workstation)	高度な処理能力が求められる技術分野やネットワークサーバ分野で利用。
パーソナルコンピュータ(PC; Personal Computer)	家庭やオフィスなどで多目的に利用されるコンピュータ。 **パソコン**ともいう。
携帯情報端末(PDA; Personal Digital Assistant)	ノートパソコンよりも小型で携帯して持ち運ぶことが可能なコンピュータ。スマートフォンによって復権している。
マイクロコンピュータ (Micro Computer)	１つのチップに納められた最も小さいコンピュータ。 家電製品などに組み込んで利用される。**マイコン**ともいう。 (マイコンはマイクロコントローラの略になることもある)

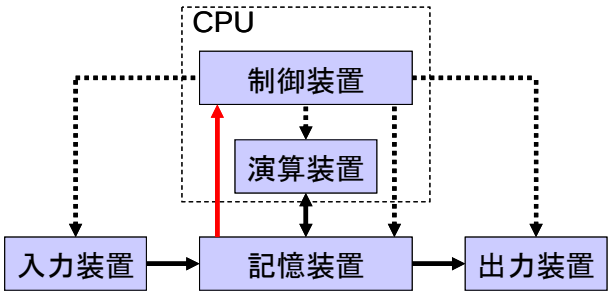
近年の半導体集積技術の向上により、単純に性能のみではコンピュータは分類できなくなってきている。例えば PC であっても WS レベルの処理が可能なものまでできている。ただし、耐故障性や故障時の代替処理の迅速化など、個人向けでは重要視されない部分において、WS は PC よりも遥かに高い能力を持っていたりする。また、PC の普及によってほぼ消滅してしまったオフィスコンピュータという区分もある。

3.1 コンピュータの構成

(1) ５大装置(機能)

コンピュータを構成する機械的・物理的な実体のことをハードウエアと呼ぶ。コンピュータを構成するハードウエアは、入力、出力、記憶、制御、演算の５大装置から構成される。この５つの装置を５つの機能と呼ぶこともある。

装置	機能	代表的機器
1.入力装置(機能)	データやプログラムを呼び込む	キーボード、マウス、タブレット
2.記憶装置(機能)	データやプログラムを記憶する。記憶装置には、主記憶装置と補助記憶装置がある。	
	主記憶装置：コンピュータ内部にあり CPU で実行するプログラムやデータを格納する。主記憶装置は揮発性（電源を切ると記録内容が消える）である。	DDR(2|3)-SDRAM Mobile DDR-SDRAM
	補助記憶装置：大容量低価格、不揮発性という特徴をもつ。	CD-ROM、DVD-ROM、BD-ROM、SSD、HDD、磁気テープ
3.出力装置(機能)	処理結果のデータを外部に取り出す。	ディスプレイ、プリンタ、スピーカー
4.演算装置(機能)	計算、比較、判断などを行う。	CPU(Central Processing Unit)：演算装置と制御装置からなる。
5.制御装置(機能)	入力装置、記憶装置、出力装置、演算装置の制御を行う。	



(実線はデータの流れ、破線は制御の流れ、赤線は命令の流れを示す。)

(2) コンピュータの基本的な構成

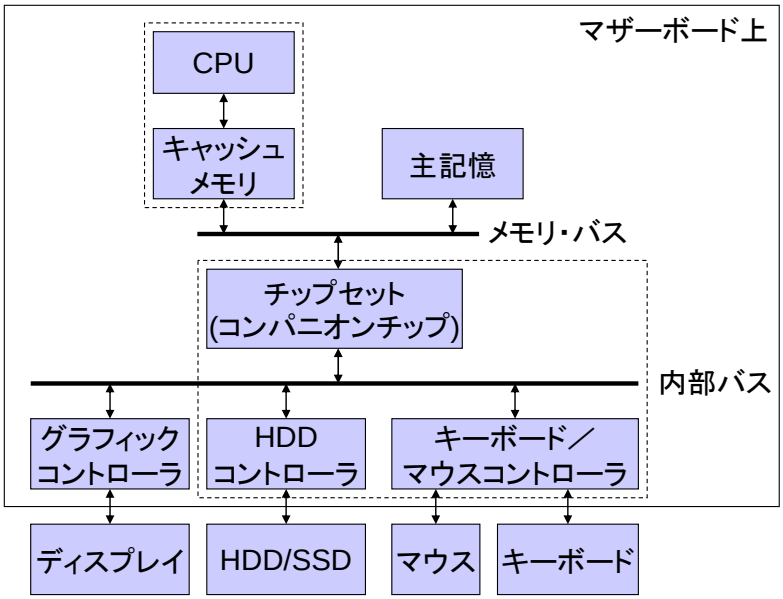
コンピュータ内部では、各装置はバスという共用の信号線を介して接続されている。以下のバス間のやりとりを行うためのチップセット(コンパニオンチップ)が準備されている。これらはマザーボードと呼ばれる基板上に実装される。

- メモリバス： CPU(+キャッシュ)とメモリを接続する。Intel 社の P4 バスなど。近年では、メモリは直接 CPU に接続される傾向にある。
- 汎用バス： 各種入出力装置、記憶装置を接続する。PCI バスなど。近年では、高速化

のためにバスではなくチップセットと１対１接続になる傾向がある(例:PCI Express)。また、入出力インタフェースもチップセットに統合される傾向にある。

・ 専用バス： グラフィックコントローラなど、特にデータ転送速度が要求される装置などのために、共用による性能低下を排除したバス。AGP バスなど。

近年では、複数の入出力インタフェースの実装による煩雑さをさけるため、共通化した入出力インタフェースを実装する傾向にある(例：USB, ATA, S-ATA など)。



3.2 半導体素子(トランジスタ)とメモリ

CPU や主記憶装置(メインメモリ)など、コンピュータの構成要素のマザーボード上のもののほとんどには、半導体素子が使用されている。この半導体素子の中でも、近年のコンピュータは**トランジスタ**を用いている。

トランジスタは半導体素子の中で最も基本的な素子。トランジスタは、アナログ信号では電流を増幅するが、コンピュータで使用するディジタル信号では、0 と 1 を切り替えるスイッチの役割をする。かつてはバイポーラ型のトランジスタなども使われていたが、現在の集積回路中では、電圧で状態を切り替える MOSFET(Metal Oxide Semiconductor Field Effect Transistor)が主流である(MOSFET はユニポーラ型トランジスタの 1 種)。

(1) 集積回路(Integrated Circuit)

集積回路(IC: Integrated Circuit)はトランジスタなどの半導体素子を集積して組み合わせて、機能を持たせた電子回路。集積回路は使用する半導体素子の種類によって、主にバイポーラ型と CMOS 型の２種類に分けられる。

バイポーラ型	バイポーラ型トランジスタを組み合わせた集積回路。回路の動作が高速であることが特徴であるが、消費電力が大きく、発熱量が多くなる欠点がある。90 年代中盤に集積回路の主流から外れたが、強い電流を出力できるという利点があるので、使われている分野もある。
CMOS(Complementary Metal Oxide Semiconductor)型	２種類の MOSFET を組み合わせた集積回路。消費電力が少ないことが特徴で、バイポーラ型よりも集積できるので安価で製造できる。このため、現在のコンピュータの多くは CMOS 型集積回路によって構成されている。

(2)IC メモリ

主記憶装置には IC メモリが使用されている。IC メモリは、ROM と RAM に分けられる。近年では集積技術の向上により、演算回路と同じシリコンチップ上に集積されたり、chip-on-chip 技術で同一パッケージに集積されていたりもする。

ROM(Read Only Memory)

読み出し専用の IC メモリ。電源を落としても内容は失われない（不揮発性）という特徴がある。近年では、フラッシュメモリ等の書き換え可能な物も存在するが、(回数は多いとはいえ)書き換え可能回数があるため、こちらに分類しておく。

マスク ROM (Mask ROM)	製造するときに記憶内容を書き込み、後から内容を書き換えることができない ROM。
PROM (Programmable ROM)	一度だけデータを書き込める ROM。一度書き込んだデータは変更できない。
EPROM (Erasable PROM)	記憶内容を紫外線で一括消去して、再度、内容を書き換えることができる ROM。
EEPROM (Electrically EPROM)	電気的にブロックまたはバイト単位で記憶内容を消去し、データを書き換えることができる ROM。
フラッシュメモリ (Flash Memory)	電気的に一括またはブロック単位でデータを消去して内容を書き換えることができる ROM。SD カードなどの記録メディアに主に使われる。近年では、SSD という形で HDD の代替に使われることもある。

RAM(Random Access Memory)

読み出しと書き込みが可能な IC メモリ。電源を落すと内容が失われる（揮発性)。

DRAM	コンデンサを使用してデータを記憶する RAM。コンデンサ中の電荷

(Dynamic RAM)	は漏れるため、一定時間ごとに再書き込み（リフレッシュ）を行い、記憶内容を保持する。SRAM よりも低速。集積化しやすく、低コスト化、大容量化に対応できるので**主記憶装置**に用いられる。
SRAM (Static RAM)	フリップフロップ回路を用いてデータを記憶する RAM。リフレッシュが不要。DRAM に比べて高速。**キャッシュメモリ**に用いられる。

近年では、上記の RAM と ROM の両方の性能を満たすことを目的として、Magnetic RAM, Phase Change RAM, Registive RAM などの開発が続けられている。

3.3 プロセッサアーキテクチャ

プロセッサアーキテクチャとは、CPU の基本構成のことである。CPU は、主記憶装置からプログラム(命令語で記述)とデータを取り出して解読・実行する。このとき、CPU が解釈し実行できる命令語を**機械語**という。

(1) 機械語の命令とアセンブリ言語

機械語の命令はビット列である。ビット列のままでは人間が理解しづらいため、機械語と 1 対 1 に対応するアセンブリ言語が存在する。アセンブリ言語の命令は、**命令部**と**アドレス部**から構成される。命令部は「処理」、アドレス部は「処理対象の格納位置」を表す。アドレス部では、複数の格納位置を示す場合もあるし、格納位置の代わりに単純な数値を示す場合もある。

機械語命令の例

ADD	**A,B**

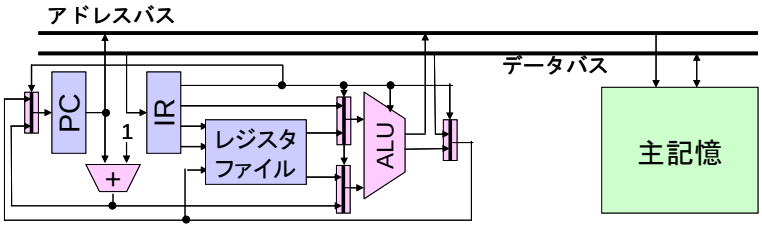
(命令部) (アドレス部)

機械語命令やアセンブリは、プロセッサの機械語の言語仕様によって異なる。この言語仕様を、**命令セットアーキテクチャ**と呼ぶ。逆に考えれば、命令セットアーキテクチャが同じであれば、CPU が異なっていても、同一の機械語命令列を受け付ける(例：Intel 社の Core-i7 と AMD 社の Phenon)。

命令セットアーキテクチャの違いにより、機械語命令の演算命令のアドレス部に**レジスタ(汎用レジスタ)**と呼ばれる CPU 内部のデータの保持場所のみを許すものと、レジスタに加えて主記憶の場所を指定できるものが存在する。前者は特に**ロード／ストアアーキテクチャ**と呼ぶ。ロード／ストアアーキテクチャでは、主記憶中のデータは一旦、レジスタにコピーされたうえ(ロード命令)、演算を適用し、レジスタに保存された結果を主記憶に書き戻す(ストア命令)形で処理される。説明が簡単になるため、以下では、ロード／ストアアーキテクチャをベースに説明する。

(2) 命令の実行

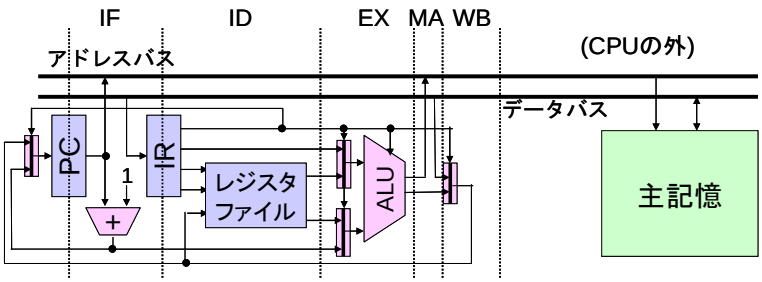
処理装置である CPU は、演算装置と制御装置から構成され、機械語命令を解読し実行する。以下の図を用いて、命令の実行プロセスを示す。



CPU 中の命令の実行は、以下の 5 段階(ステージ)に分けて考えることが多い。

1. **命令フェッチ(IF**: Instruction Fetch):
   主記憶から PC(Program Counter: 実行中の命令の位置を示す)の値をもとに機械語命令を読み出してくる。読み出してきた機械語命令は IR(Instructoin Register)に保存。命令を読み出してきたら、次の命令の読み出しに備え、PC の値を PC+1 に更新する。
2. **命令デコード(ID**: Instruction Decode):
   命令の解釈(decode)を行い、演算対象となる情報をレジスタファイルより取り出す。また、演算の指定、演算に使う値の機械語命令からの切り出し、値の ALU への送り込みのための配線の切り替え情報の作成なども行う。
3. **実行(EX**: EXecution):
   ALU(Arithmetic Logic Unit)にて演算を行う。演算は基本的にレジスタファイルより取り出した値を用いるが、分岐命令の場合は、PC の値に対して行う。
4. **メモリ(主記憶)アクセス(MA**: Memory Access):
   命令がロード/ストア命令であった場合、メモリ(主記憶)アクセスという主記憶の読み書き動作を行う。他の命令あった場合は、何も行わない。
5. **ライトバック(WB**: Write Back):
   演算結果やロードした値をレジスタファイルに書き戻す。また、分岐命令の場合は演算した PC の値を PC に書き戻す。

以下は最初の図をステージごとに分割した図である。

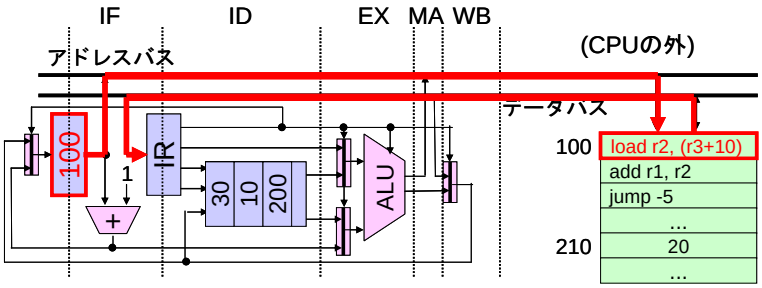


以下、上記の図をもとに、代表的な命令である演算命令、メモリアクセス命令(ロード／ストア命令)、制御命令(分岐命令)の動作を説明する。

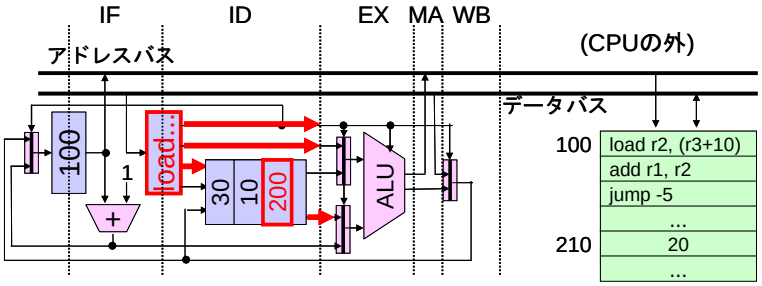
(a) メモリアクセス命令(ロード／ストア命令)

メモリアクセス命令では、レジスタの値や命令語の中の即ちから生成される実効アドレス(次節参照)で示される主記憶中のデータと、レジスタの値の入れ替えの操作を行う。以下では、主記憶中のデータのコピーをレジスタに書き込む、ロード命令を例に取って説明する。

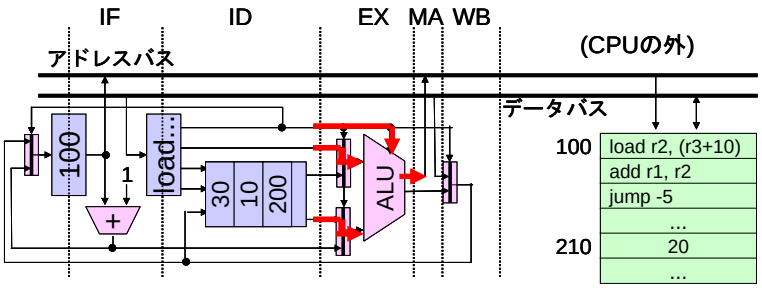
IF: PC の値を主記憶に送り、対応する位置にある命令を読み出す。



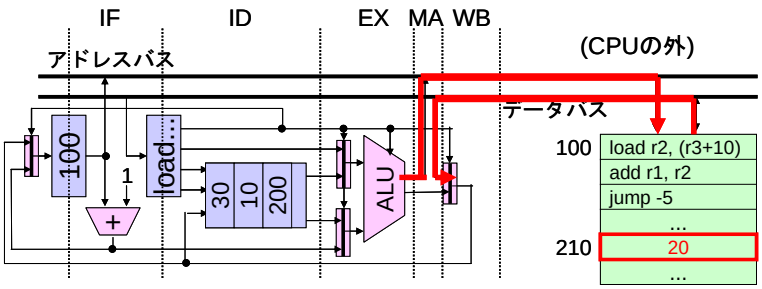
ID: 主記憶から読み出した命令語は IR に保存される。IR の内容に従い、レジスタ値を読み出して ALU に送り込んだり、命令語の一部を切り出して ALU に送り込んだり、演算を指定したりする。



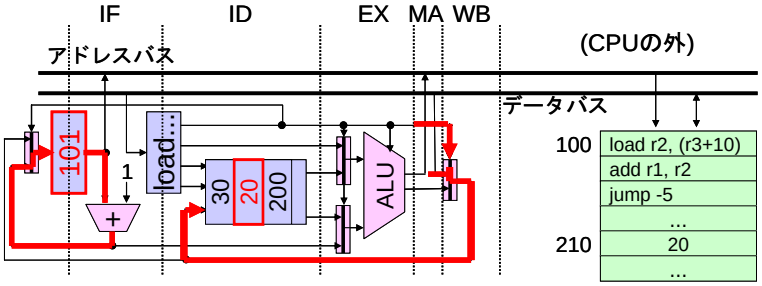
EX: ALU で加算を行い、メモリアクセスに必要な実効アドレス(次節参照)を生成する。



MA: 実効アドレスを主記憶に送り、データを読み出す。



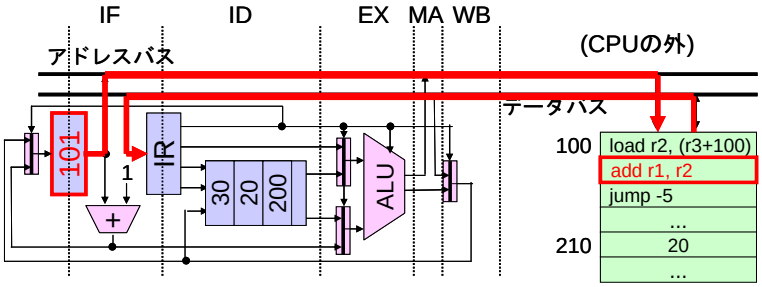
WB: 主記憶から読み出した結果をレジスタファイルに保存する。また、次の命令の読み出しに備え、PC の値を 1 増やす。



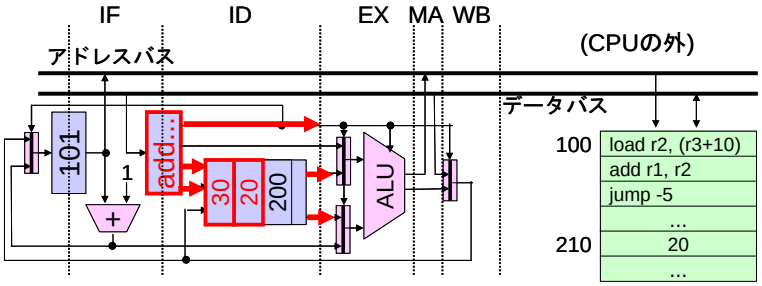
(b) 演算命令

演算命令はレジスタ中の値や主記憶中の値を演算し、書き戻す命令である。例示するアーキテクチャはロード／ストアアーキテクチャのため、演算はレジスタ中の値のみに行う。CPU の状態は、(a)のロード命令の実行が終わった状態になっている。以下、ロードしたデータを他のレジスタに加算する演算命令を例に取って説明する。

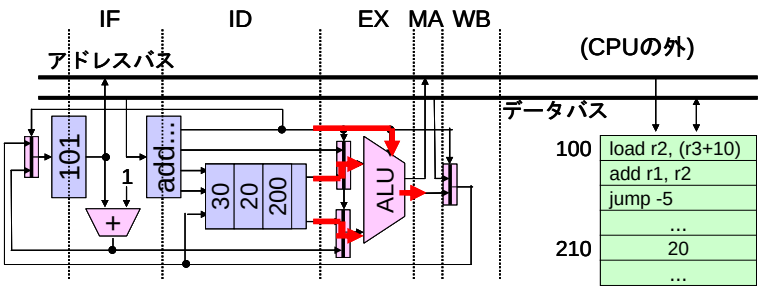
IF: (a)の時に同じ。



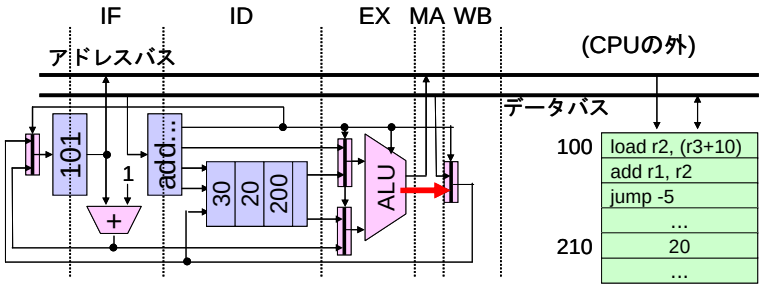
ID: (a)の時に同じだが、読み出すレジスタ値が 2 つという点が異なる。



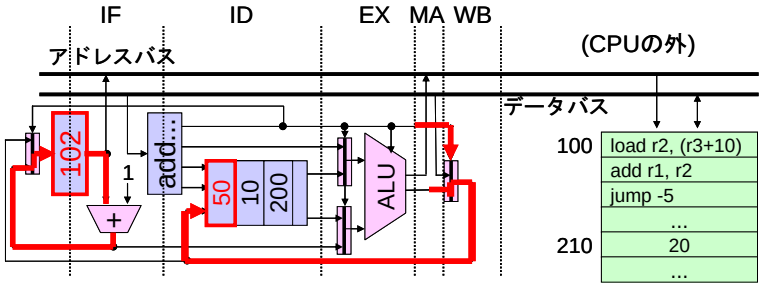
EX: 演算を行う。



MA: 何も行わない。



WB: 演算結果をレジスタファイルに保存する。また、PC を+1 する。

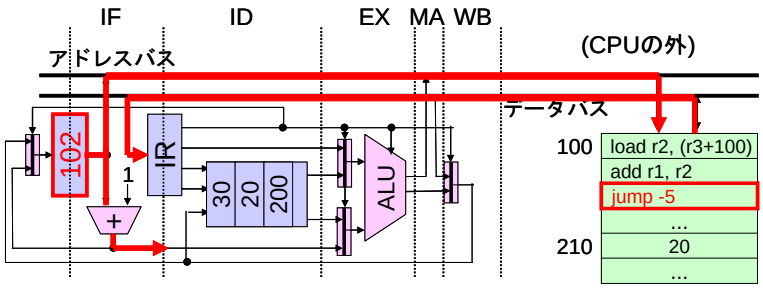


Intel 社の x86 アーキテクチャでは主記憶中の値の演算を直接用いる演算命令も許す。この場合、EX の前に主記憶中の値を読み出す MA が挿入されると同時に、EX の後にも主記憶に演算結果を書き戻す MA が残る。もっとも、現在の x86 アーキテクチャでは高速化のため、主記憶中の値を演算に直接用いる演算命令は、レジスタに値を読み込むロード命令、レジスタの値を用いた演算命令、レジスタの値を主記憶に書き戻すストア命令に分割されて実行されることが多い。

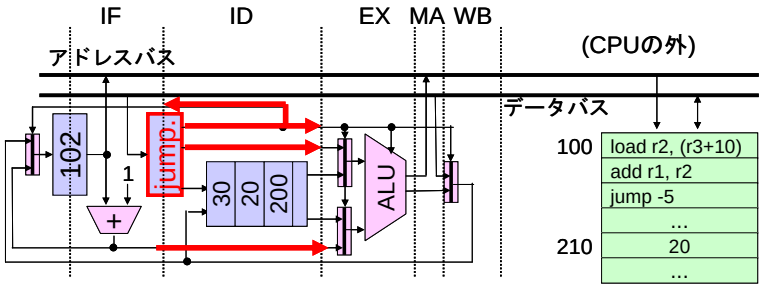
(c) 制御命令(分岐命令)

意味のあるプログラムを構成するためには、様々な条件によって処理の流れ(制御フロー)を変える必要がある。このような命令を制御命令もしくは分岐命令と呼び、プログラムカウンタの値を書き換えることで、実行する命令列を変更し、処理の流れを変える。以下、ジャンプ命令(無条件に PC を書き換える)の動作を例に取って示す。

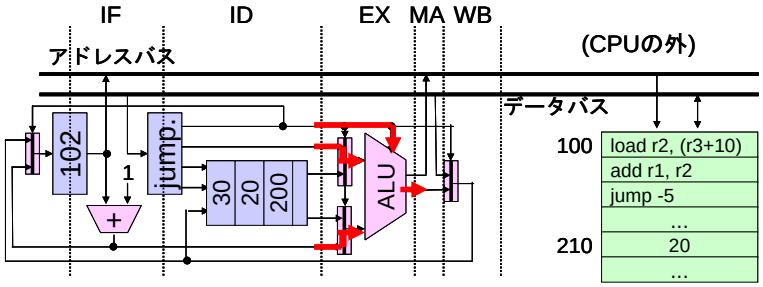
IF: (a),(b)に同じ。



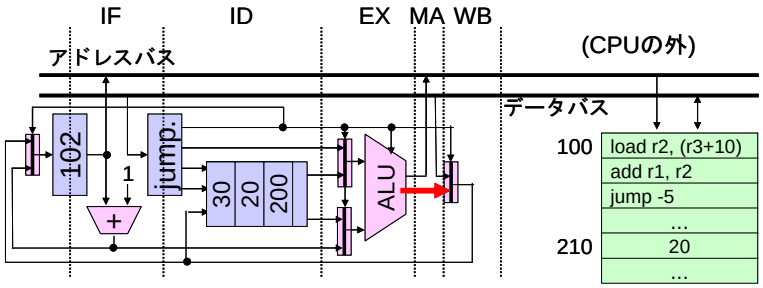
ID: 分岐命令は演算に PC+1 と命令語の一部を使うため、それらを ALU に送る。



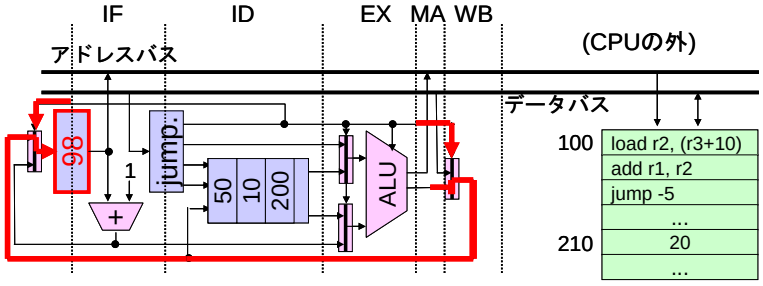
EX: ALU で分岐先アドレス(PC+1+命令語の一部)を計算する。



MA: 何も行わない。



WB: 分岐先アドレスを PC に書き込む。



実際のプログラムの記述においては、このような**無条件分岐命令**ではなく、条件によって分岐するかしないかを決める**条件分岐命令**が重要となる。分岐の条件には、値を比較した結果が 0 であった、大きかった、小さかった、などが用いられる。

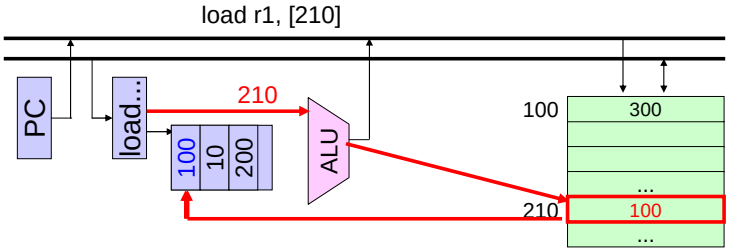
(3) アドレス指定方式

アドレスとは主記憶中の住所のようなものであり、実行すべき命令や処理すべきデータの位置の指定に用いられる。プログラムの記述を用意にするため、処理対象が格納されて

いる**実効アドレス**（**有効アドレス**）を求める方法(**アドレッシングモード**)がいくつも準備されている。以下によく使われるアドレッシングモードを示す。

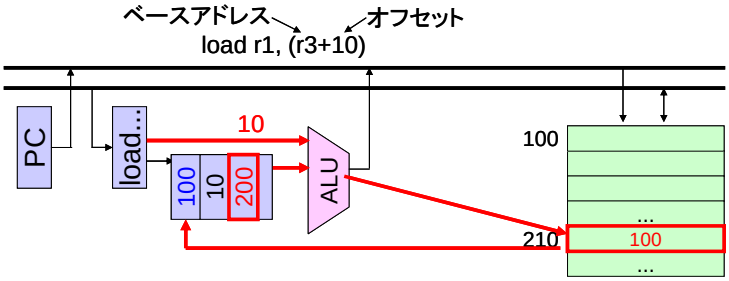
(a) 直接アドレス指定方式

命令語のビット列の一部が、そのまま実効アドレスになる指定方法。命令語の一部であらわせる範囲しか指定できない。



(b) オフセット付きレジスタ間接方式

ベースアドレス(レジスタ値)＋オフセット値(命令語の中)の結果が実効アドレスになる指定方法。一番よく用いられる。配列のアクセス等には、ベースアドレスに配列要素を示す変数(例：n)、オフセット値に変数からのオフセット(例："n+1"の"+1"の部分)を使うのが便利である。



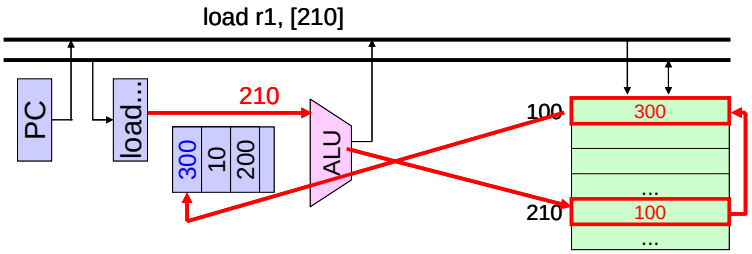
一般的にベースレジスタには汎用レジスタが用いられるが、インデクスレジスタなどの専用レジスタを用いるアーキテクチャも存在する。専用レジスタは汎用レジスタとは分かれているアーキテクチャも存在するし、汎用レジスタの一部を用いるアーキテクチャも存在する。代表的な専用レジスタとして、グローバルベースレジスタ、スタックポインタ、リンクレジスタ、などがある。

派生として、オフセットの無い単純なレジスタ間接方式や、もう 1 つのレジスタを加算するもの(レジスタ値＋(インデクス)レジスタ値＋オフセット値)も存在する。

(c) メモリ間接指定方式

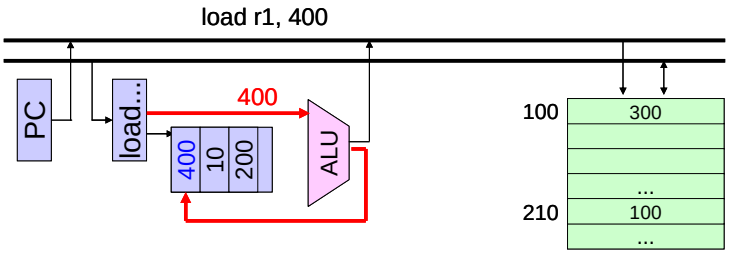
レジスタ間接指定方式などで示されたアドレスに入っているデータを実効アドレスとし、再びメモリアクセスを行った結果を読み書きするもの。ライブラリ関数呼び出し表(ジャンプテーブル)などの参照に用いられることがあるが、動作の単純化のために実装していない

アーキテクチャも存在する。



(d) 即値オペランド指定方式

命令語の一部を処理対象とするデータとする方式。メモリアクセスは行わず、命令語の一部を直接演算に用いたりレジスタに格納したりする。



(4) CPU の性能指標

異なる CPU の性能を比較するために、以下のような指標がある。

クロック周波数	CPU などの処理装置は、コンピュータ内部にある発振器から発せられる信号に同期して動作する。この発振器が「一秒間に何回発振しているか」と言う指標が、クロック周波数である。クロック周波数が 1GHz であれば、一秒間に 10 億回の信号の振幅が発生することを意味する。同じ設計ならば、クロック周波数が高い方が性能が高くなる。
CPI (Clock cycles Per Instruction)	CPI とは、1 命令を実行するのに要するクロックサイクル数（クロックの振幅の数）のことをいう。例えば、Intel 社の Pentium 4 の場合、1 命令の実行に 4～30 クロックサイクル程度を必要とする。命令によって、CPI は異なる(興味があるならば、Intel 社の開発者用ドキュメント参照)。後述するパイプライン化などをしていたりすると、処理が並列に行われているため、多くの命令を実行した時の平均の CPI は単一の命令を実行した時よりも小さくなる。
MIPS(Million Instructions Per Second)	1 秒間に実行できる命令の数を百万単位で表したもの。 25MIPS = 25,000,000 命令を 1 秒間に実行
FLOPS(FLoting oint Operations	1 秒間に実行できる浮動小数点演算(倍精度浮動小数点)の数を表したもの。スーパーコンピュータの性能指標によく使われる。ピーク性能

Per Second)	(理論最大値)と実効性能(プログラムを動作させて測定した値)に注意。

上記の指標をもとに、あるプログラムを実行した時の CPU 間の性能の違いや予想される処理時間を見積もったりすることができる。以下は、その例である。

- 1.0×10^{15} 個の行列演算を行う科学技術計算を 1.0×10^{12}FLOPS(1TFLOPS)のスーパーコンピュータで実行する時の実行時間。(ただし、演算に付随する処理は取るに足らないほど短時間で終了すると考える)

  $1.0\times10^{15} \div 10\times10^{12} = 1000$ 秒

- 2GHz で動作している CPU であるプログラムを動作させた時の平均 CPI が 0.5 の場合、1 秒あたりに実行される命令数。

  $2.0\times10^{9} \div 0.5 = 4\times10^{9}$ 命令／秒

- 800MHz で動作する CPU の平均 CPI が演算命令で 1、メモリアクセス命令で 5、分岐命令で 10 である。演算命令 50%、メモリアクセス命令 30%、分岐命令 20%のプログラムを実行した時の 1 秒あたりに実行される命令数は？

  平均 CPI は $1\times0.5 + 5\times0.3 + 10\times0.2 = 4$

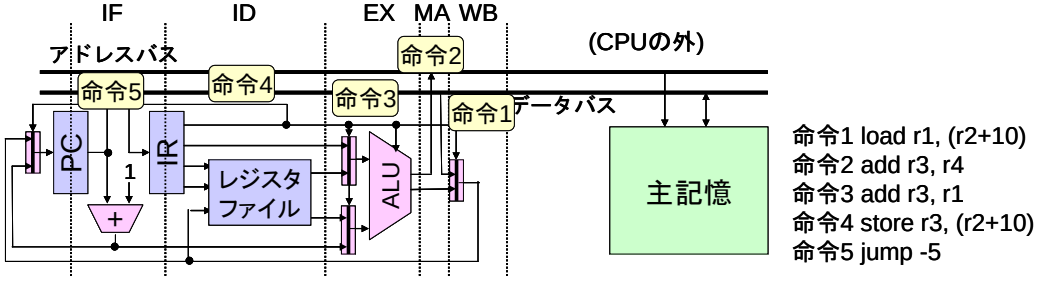
  $800\times10^{6} \div 4 = 200\times10^{6}$ 命令／秒

(5) CPU の高速化技術

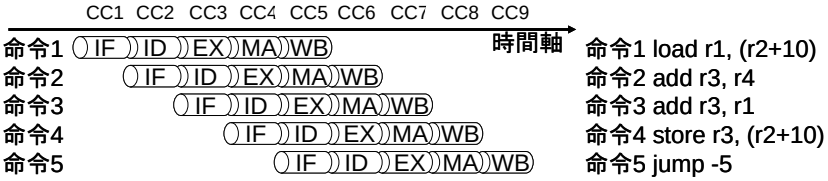
CPU の高速化はプログラムの実行の高速化につながるため、昔から様々な手法が提案されている。これらの技術により、近年の高性能 CPU では CPI は 0.5 以下(秒間 2 命令以上を実行)までになっている。以下では、その一部の概要を説明する。

(a) パイプライン処理

CPU で命令の実行動作をいくつかの**ステージ**に分割して、ステージ単位で並列処理を行う技術を**パイプライン処理**と呼ぶ。前述の IF, ID, EX, MA, WB を個々のステージとして、それぞれを並列に動かすものと考えると分かりやすい。下の図の例では、命令 1 が WB で結果をレジスタファイルに書き込むのと平行して、命令 2 が MA の処理を行い(実際は処理なし)、命令 3 が EX で演算を行い、命令 4 が ID でレジスタ読み出しなどを行い、命令 5 が IF で主記憶から読み出されている。この動作は、ベルトコンベア上の流れ作業を考えると理解しやすい。



パイプライン処理の様子は、以下の図のように時間軸を使って表されたりもする。この時の時間軸の単位は通常クロックサイクル(CC)となる。下の図で CC5 の時を見ると、上の図の状態となっていることが見て取れる。

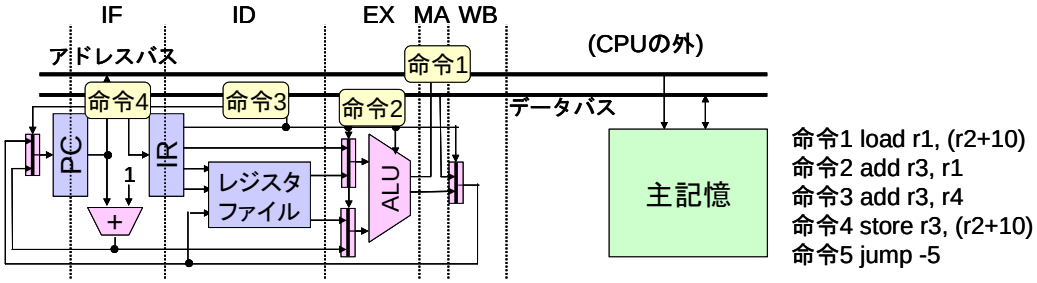


ステージ数は CPU のアーキテクチャによってことなる。パイプライン処理のステージをさらに細分化したものを、**スーパーパイプライン**という。最も細分化された物の一例として、Intel 社の Pentium 4 の 31 ステージへの細分化がある。

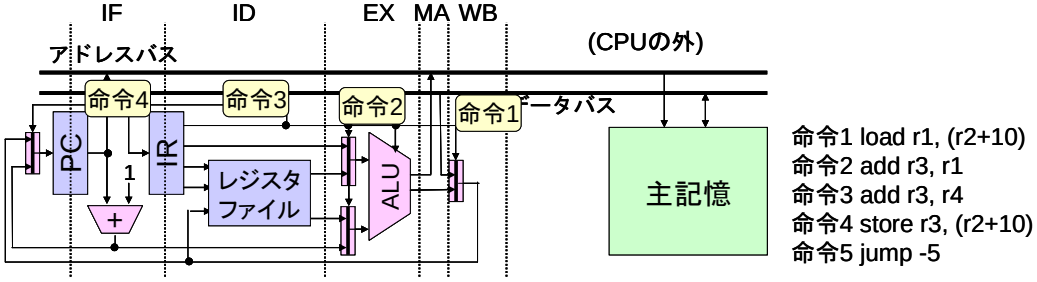
理想的なパイプライン化された実行は全てのパイプラインステージで別個の命令の処理が進んでいる状態だが、データハザード、制御ハザード、構造ハザードによってそれが妨げられる。

データハザード

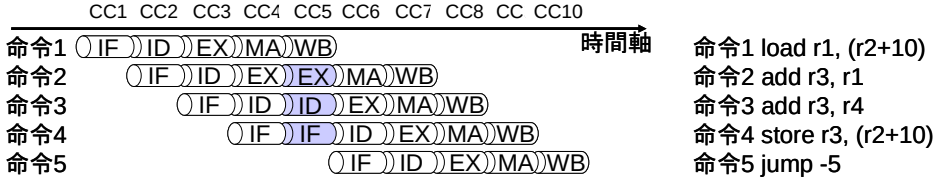
下記のように、命令 1 でロードした値を命令 2 で演算しようとした場合、命令 2 が EX ステージに来たときにはまだ値は MA ステージで読み出す途中である。このように、演算しようとする値が使えないことによるハザードを**データハザード**と呼ぶ。



この場合、下記の図のように、命令 2 以降は命令 1 が主記憶アクセスを終了するまで各ステージで待つことになる。この待ち合わせによるパイプライン処理の停止を、**パイプラインストール**と呼ぶ。また、下記の図では主記憶から読み出した値を迅速に使うため、値をレジスタファイルに書き込むのと平行して ALU に送り込んでいる。これにより、レジスタファイルを介して値を渡すよりもパイプラインストールの量が減る。このような値の受け渡しを**データフォワーディング**と呼ぶ。データフォワーディングの経路は MA ステージから EX ステージのみならず、EX ステージから EX ステージのルートも存在する。

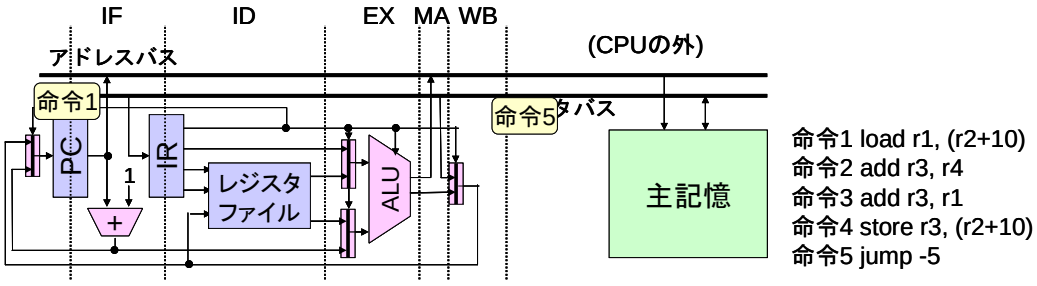


パイプラインストールを時間軸で現すと、以下の図のようになる。待ち合わせは、それぞれの命令が直前にいたステージで行われることに注意されたい(図の色塗り部)。

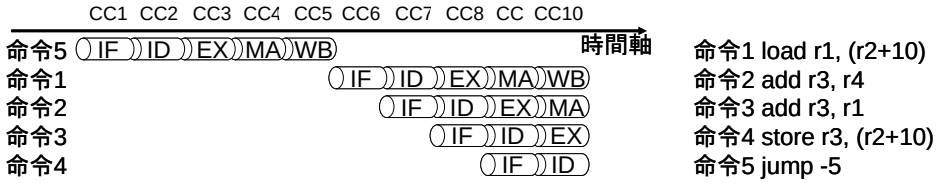


制御ハザード

下記の命令列で命令 5 から命令 1 に戻る場合、命令 5 の実行が完了して命令 1 を示す値を PC に書き込まない限り、命令 1 の主記憶からの読み出しはできない。このようなことによるハザードを**制御ハザード**と呼ぶ。



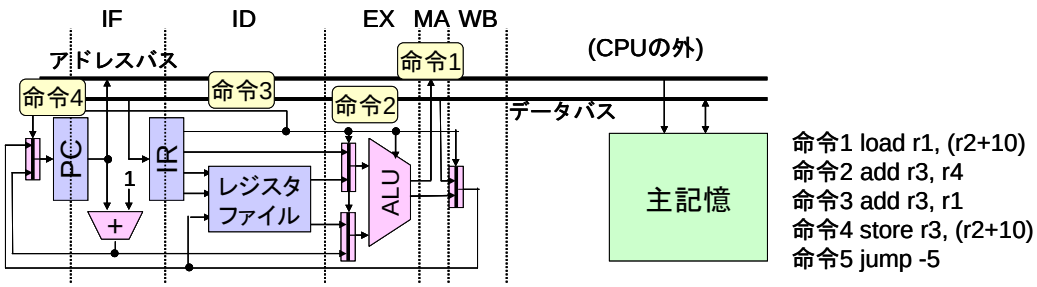
この時の時間軸を用いた図は以下のようになる。



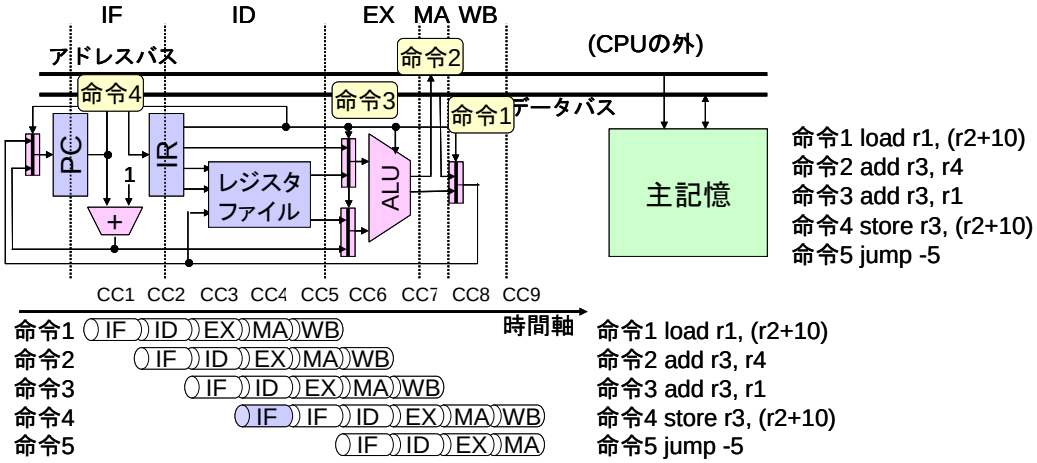
制御ハザードによるパイプラインストールは長いので、**分岐予測**という技術を使って先に命令を読み出す手段がよく用いられる。

構造ハザード

下記の構造では、命令 1 のロード処理による主記憶読み出しと命令 4 の命令語の主記憶からの読み出し(命令フェッチ)は主記憶という資源を同時に用いる。もし、主記憶が同時に 1 つまでの読み書きしか受け付けない場合、どちらかの主記憶読み出しの要求は後回しにされる。このようなハザードは**構造ハザード**と呼ばれる。



上記の場合では、通常、先行する命令 1 の要求を優先する。そのため、命令 4 の読み出しは待たされることになる。



上記のような構造ハザードは、主記憶を一度に 2 つの読み書きを許容するように設計するか、命令専用の主記憶とデータ専用の主記憶に分割する(古典的なハーバードアーキテクチャ)ことで無くすことができる。

(b) RISC と CISC

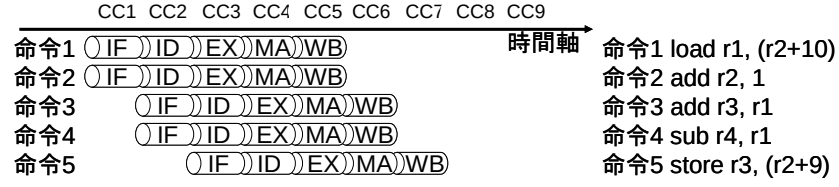
RISC(Reduced Instruction Set Computer)とは、単純な命令セットだけをハードウエアで用意し、構造を単純化したコンピュータ。命令の処理が少なくなり、プログラム中の命令が多くなる。一つの機械語命令を１クロックサイクルで実行できるように揃えることで、パイプライン処理を効果的に行う。

CISC(Complex Instruction Set Computer)は、一つの機械語命令で複雑な動作を行う命令セットを、マイクロプログラムで制御するコンピュータをいう。命令ごとの実行時間が異なるため、パイプライン処理には不向きである。

しかしながら、近年では、CISC の CPU でも内部で複雑な命令を RISC のような単純な命令に分解して処理することがよく行われており、ハードウェア設計の観点からは RISC と CISC の優劣はつけがたくなっている。

(c) スーパスカラ

命令を実行するユニットを複数搭載することで同時に実行できる命令数を増やした技術。ただし、同時に実行しようとする命令の間でハザードの要因があると、同時に実行できない。このハザードの検出は、プロセッサ自身が行う。

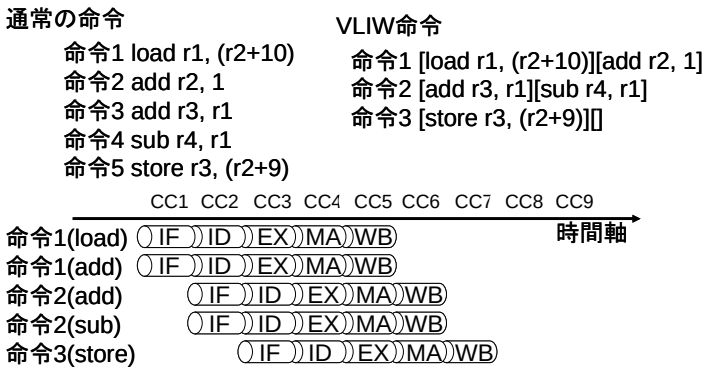


上記の例では、2 命令のスーパスカラ実行により、(a)では実行に 9 クロックサイクルかかっていた命令列が 7 クロックサイクルで完了している。

近年の高性能 CPU では、4 命令スーパスカラ実行あたりまで実現している。ただし、このように多数の命令の並列実行を行うためには、より多数の命令から、命令の並び順を無視して実行する、**アウトオブオーダ処理**の併用も必要となる。

(d) VLIW(Very Long Instruction Word)

命令語長を長くすることにより 1 命令で複数の演算処理を指定し、演算の並列に実行を可能にした技術。以下では、1 命令中に 2 つの演算を指定できる、2 演算の VLIW を例に示す。



(c)のスーパスカラと同様に、命令列の完了が 7 クロックサイクルに短縮されている。

CPU 自身が並列実行可能かどうかを判定しなければならないスーパスカラとは異なり、VLIW ではコンパイル時に、コンパイラが同時に実行可能な複数の演算を組み合わせて、一つの命令とする。CPU は VLIW 命令をそのまま複数の実行ユニットに渡すので、ハードウエアの構造が単純になり、動作周波数をあげやすいという利点がある。

(e) マルチプロセッサ

複数の CPU を搭載し、処理を分散させることにより処理性能を向上させる技術。マルチプロセッサは、主記憶装置を共有するか否かで密結合と疎結合に分類される。

密結合マルチプロセッサ	各 CPU が一つの主記憶装置を共有し、一つのオペレーティングシステムの下に稼動する方式。
疎結合マルチプロセッサ	各 CPU はそれぞれ主記憶装置をもち、オペレーティングシステムも別々で起動することができる方式。
マルチコアプロセッサ	1 つのシリコンチップの上に、複数の CPU を搭載した構成。密結合マルチプロセッサの 1 形態。近年の半導体集積密度向上により、この実装が増えている。