# 第5章 C6000 DSPシリーズのコア詳細

本章では，C6000 DSPシリーズの5種類あるコアについて，それぞれの特徴と命令セットを解説します．この章の内容は知らずともシステム開発は可能です．それはCコンパイラが「自動的に」最適化してくれる上，最適化のためのアドバイス・ツールもあるからです．しかし，コアの仕組みと命令セットを知識として知っておいた方が，ソフトウェアの最適化をする際に役立つはずです．

## 5-1 C6000シリーズ：五つのコア

C6000シリーズには5種類のコアがあり，それぞれの概要は**図5-1**のとおりです．

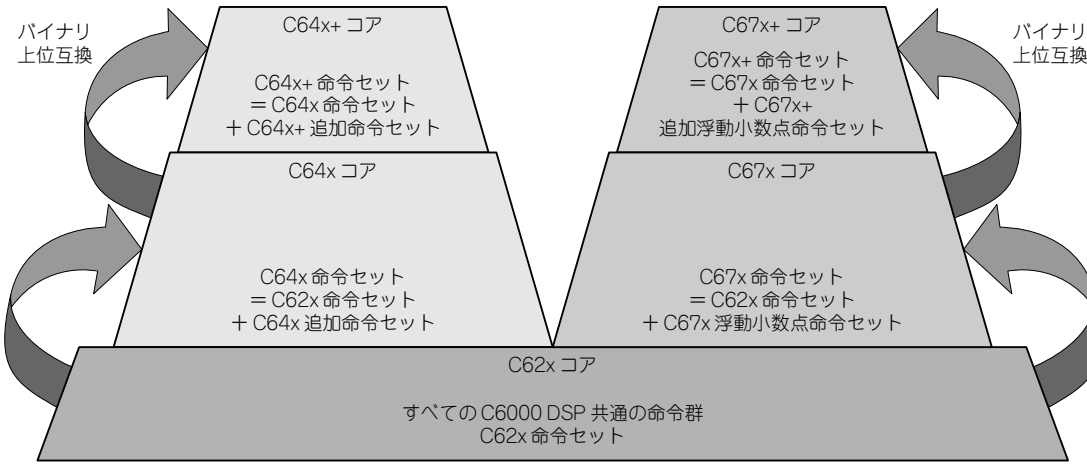


**図5-1　C6000シリーズの5種類のコアの関係**

### ● C62xコア

C62xコアはC6000シリーズの基本コアであり，C62x命令はすべてのコア共通の命令セットとなっています．1サイクルに2回の16ビット積和演算が実行できます．

### ● C64xコア

C62xコアの上位互換で，16ビット積和演算能力が2倍(1サイクル4回)になり，高速に画像処理演算をするために，8ビット加減乗算命令などが追加されています．8ビットの積和演算では，1サイクルに8回の演算が実行できます．レジスタ数とデータ幅はC62xの2倍になっています．

### ● C64x+コア

C64xコアの上位互換です．C64xと比較し，16ビット積和演算能力が2倍になっているほか，メモリ保護機能や特権モードなどのRISC機能が取り込まれています．1サイクルに2回の32ビット積和演算，もしくは，1サイクルに8回の16ビット積和演算が実行できます．

### ● C67xコア

C62xコアを基とし，浮動小数点演算を実行できるようにしたコアです．1サイクルに2回の32ビット単精度浮動小数点積和演算，64ビット倍精度浮動小数点演算が実行できます．

### ● C67x+コア

C67xコアの上位互換コアです．32ビット浮動小数点加減乗算が1サイクルに6回演算できます．C62x/C67xと比較して，レジスタ数は2倍です．

＊　　＊

**図5-1**にあるように，C64x+コアはC64xコアのバイナリ上位互換で，C64x/C64x+コアはC62xコアのバイナリ上位互換です．また同じように，C67x+コアはC67xコアのバイナリ上位互換で，C67x/C67x+コアはC62xコアのバイナリ上位互換です．

## 5-2 一般的なプロセッサの動作は？

C6000 DSPシリーズの基本であるC62xコアの説明の前に，C6000コアの詳細を理解するために必要なプロセッサについて最低限の説明をします．

まず，一般的なマイクロプロセッサは，次のような動作を繰り返し行います．

フェッチ：命令をプログラム・メモリからコアに取り込む
デコード：取り込んだ命令を解釈する
実　　行：解釈したことを実行する

8ビット・マイクロプロセッサの時代は，これらの3ステージ(フェッチ/デコード/実行)を順番に実行していました．すると，**図5-2(a)**のように，1命令当たり3サイクルかかります．そこで，動作周

波数を上げるために，パイプラインという手法が用いられています．パイプライン処理とは，**図5-2(b)**にあるように，デコードしているときに次の命令をフェッチし，オーバラップしながら処理していきます．この場合，1/3の時間で処理できます．

このパイプライン処理は，よく車の組み立て工場にたとえられます．どういうことかというと，まずフェッチ機構という機械がプログラム・メモリから命令を取り出し，ベルト・コンベアに載せることだけを行います．そしてデコード機構ではベルト・コンベアに載っている命令を解釈し，解釈した内容を再度ベルト・コンベアに載せることだけを行います．さらに実行機構と呼ばれる機械では，その流れてきた解釈内容を実行し続けるのです．

フェッチ機構，デコード機構，実行機構のそれぞれは，流れ作業で動いています．そのため，一定時間に命令を実行する数は非常に多くなります．

しかし，このようにパイプライン処理をしているプロセッサでは，分岐命令の実行に時間がかかります．それは分岐命令を実行機構で実行すると，フェッチ/デコード機構にある命令Bと命令Cの実行

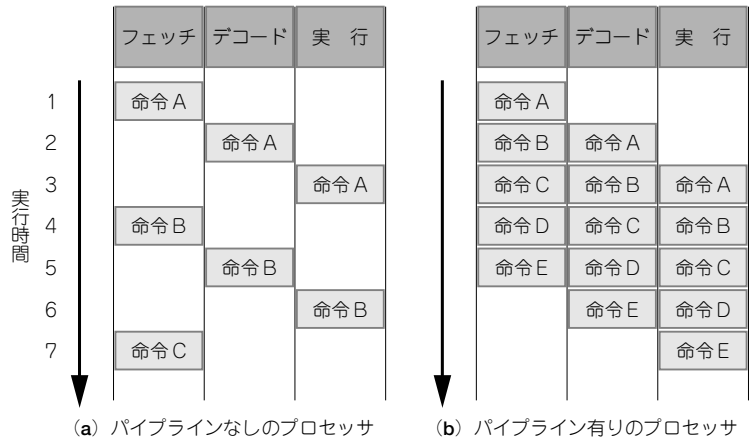


**図5-2　パイプライン処理**

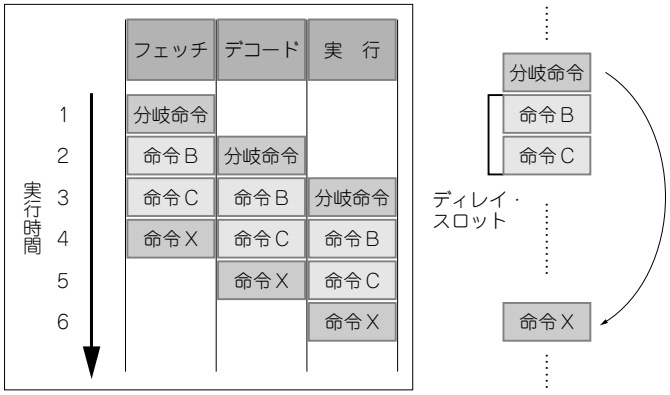


**図5-3　ディレイ・スロットの図**

をキャンセルして，分岐先の命令Xのフェッチから実行を始める必要があるからです．つまり，流れ作業ができなくなってしまい，この例では，分岐命令は3サイクルかかることになります．

これを回避するため，分岐命令後の二つの命令(命令Bと命令C)の実行をキャンセルせず，そのまま実行しています．この命令のことを遅延分岐命令，命令Bと命令Cをディレイ・スロットにある命令という言い方をします(**図5-3**)．

## 5-3 VelociTIアーキテクチャ vs. VLIWアーキテクチャ

C62xコアは，VLIWをベースに少し改良したアーキテクチャを採用しています．このアーキテクチャをVelociTIと呼びます．ここでは例を挙げながら，従来の典型的なVLIWとVelociTIとの違いを解説します．

では，次のようなアセンブラ・プログラムを実行することを考えましょう．

```
        B
||      SUB
||      ADD
        ADD
||      MPY
        MPY
||      LDH
||      LDH
```

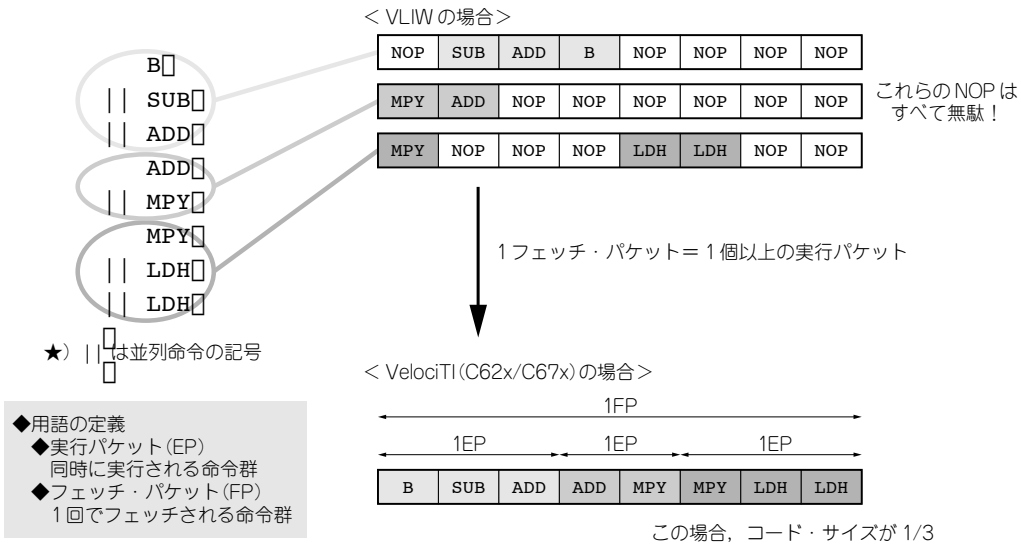


**図5-4　VelociTIアーキテクチャ vs. VLIWアーキテクチャ**

ここで，ADDやSUB，Bなどは命令を，それらの命令の前にある「||」の記号は前の命令と同時に実行することを示しています．この場合，最初にB命令とSUB命令，ADD命令を同時実行します．また，同時に実行する命令群を実行パケット(EP)，一度にフェッチする命令群をフェッチ・パケット(FP)と呼びます．

従来のVLIWの場合，同時に実行する命令群(EP)と命令を実行する機能ユニットが1対1で対応しています．何も実行しないユニットに対してはNOP命令を入れる必要があります．つまり，8個の機能ユニットがあるVLIWの場合，最初のB命令とSUB命令，ADD命令以外はNOP命令で埋める必要があります．すると，どうしてもコード・サイズが大きくなってしまいます．これは，従来のVLIWの欠点といわれていました(**図5-4**)．

そこで，VelociTIアーキテクチャでは，コード・サイズを削減するために余分なNOP命令を削除し，8命令単位にまとめた形(1FP)で一度にフェッチしています．この例では，コード・サイズが1/3になります．そして，実際の実行では，最初にB||SUB||ADD命令，次にADD||MPY命令，最後にMPY||LDH||LDH命令と，順番にタイミングをずらして機能ユニットに渡します．このようにして，VelociTIアーキテクチャではコード・サイズの問題を回避しています．

## 5-4　C62xコア動作

C62xコアの動作は次のような流れになっています．

フェッチ → ディスパッチャ → デコード → 実行

今まで述べたように，機能ユニットを8個搭載しているので，フェッチでは32ビット長の命令を同時に8個取り込みます．このフェッチを見かけ上1サイクルで行うため，プログラム用メモリとコアの間を，32ビット×8＝256ビット幅で接続しています．フェッチ後，タイミングをずらして機能ユニットに渡します．これを「命令のディスパッチ」と呼びます．タイミングを割り振られた命令は，各機能ユニットで命令のデコードもしくは実行をします(**図5-5**)．

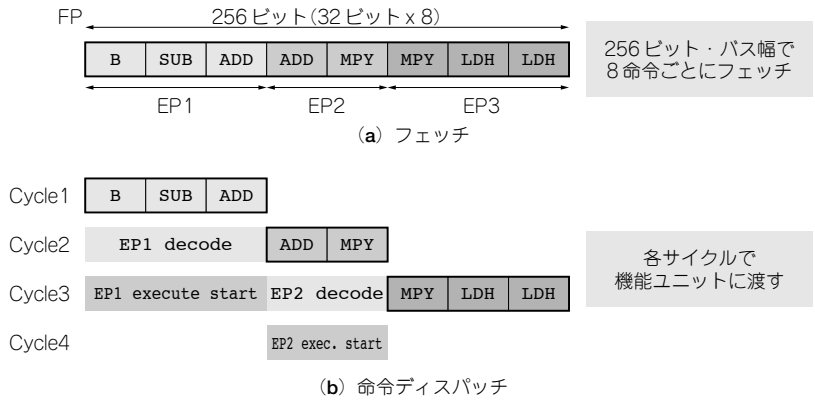


**図5-5　命令フェッチとディスパッチ**

実行のために，.L/.M/.S/.Dユニットと呼ばれる4種類のユニットがA側とB側で2セット用意されています．A側の機能ユニット(.L1/.M1/.S1/.D1)で実行する各命令のソースやデスティネーション・レジスタは，16個の32ビット・レジスタA群(A0～A15)から選択します．また，B側の機能ユニット(.L2/.M2/.S2/.D2)で実行する各命令のソースやデスティネーション・レジスタは，16個の32ビット・レジスタB群(B0～B15)から選択します．通常の使い方では，A側/B側内のレジスタ使用に関しての制限はありません．そのためCコンパイラも効率の良いプログラムを生成しやすくなっています．

しかし，A側とB側間でデータのやり取りができないと困ります．このため，1サイクルに1回ずつ，レジスタB群からA側のユニットへ，またレジスタA群からB側のユニットへ，実行する命令のソー

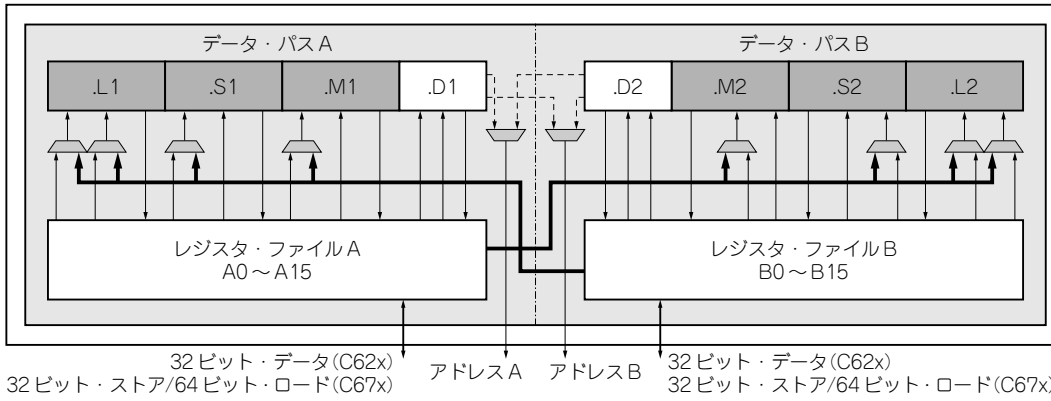


**図5-6　C62x/C67xコアの詳細な図**

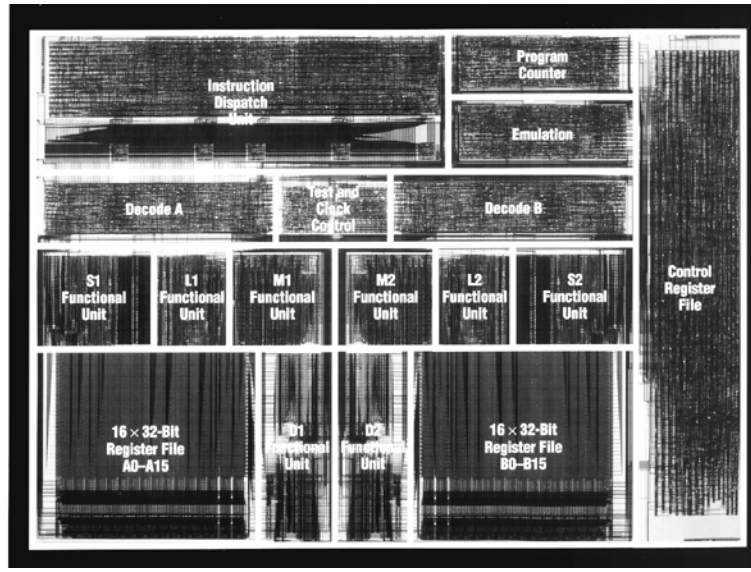


**図5-7　実際のC62xコア**