第7章 半導体プロセスの進化とCPUの歴史を見ながら

組み込み向けの最適CPU設計を考える

杉本 英樹 Hideki Sugimoto

これまで商用CPUを開発する各社は，さまざまな命令セット・アーキテクチャ（Instruction Set Architecture; 以下ISA）や，マイクロアーキテクチャ（MicroArchitecture）を持つCPUを開発し，その改良にしのぎを削ってきました．本稿では，CPUの進化の歴史を振り返りながら，現在のプロセス・テクノロジを踏まえた上で，用途に合わせた最適なCPUの構成を考察します．

半導体プロセスとCPUの歴史

●CPU高速化の変遷

▶初期…興味の中心は回路の使いまわし

初期のCPU（LSI内に構成されたものを対象にする）はアキュムレータと演算器（主にArithmetic Logic Unit；以降ALU），それらを接続するバスなど，少数の資源で構成されていました（**図1**）．

搭載できるトランジスタ数に制約があったため，トランジスタをいかに効率よく使用するかが性能を出すための要であり，いかに共通的な要素でCPUを構成するかがマイクロアーキテクチャ設計の興味の中心でした．ほぼ全ての命令でほぼ全ての資源が使用されていることが，**図1**を見ても容易に分かると思います．

▶トランジスタ数増大に伴い高速化に回路面積を割くように

時代が少し進むと，搭載できるトランジスタ数に余裕が出始め，性能のボトルネックとなる処理を改善するためにその余裕を使うことになります．例えば，外部からの命令フェッチの回数を減らすために1命令でより複雑な処理ができるようにしたり，外部とのデータのやり取りを減らすためにレジスタの本数を増やしたり，CPU内でのデータ転送（レジスタ−ALU間など）時間を短縮するために内部バスの本数を増加させたりといったようにです．

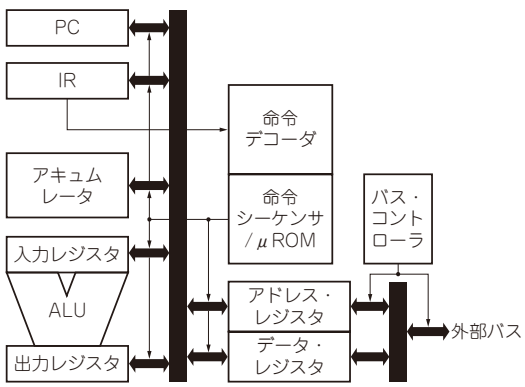

図1 初期のCPUでは各命令が回路を使いまわすので，全体の構成がシンプルだった
PCはプログラム・カウンタを，IRは命令読み出しを表す．

表1 プロセス・テクノロジの進化と各世代のCPUアーキテクチャ
RISC成長期以降では単体CPUにとって十分すぎる量のトランジスタが利用できるようになった
IPCはクロック当たりの命令実行数，OoOはアウト・オブ・オーダ実行を表す．解説は本文参照

世代	初期CISC	CISC全盛	初期RISC	RISC成長	並列化拡大
代表的なプロセス	1μm	0.5μm	0.25μm	32n～130nm	16nm以下
トランジスタ数	～1万	～10万	～1000万	−	
ISA	CISC（100命令種以下）	CISC（100命令種以上）	RISC		
パイプライン段数	2以下	3～5	5～10	10～20	15～30
命令同時発行数	1			～4	～8
IPC	0.25程度	0.5程度	約1	～2	～4
動作周波数［Hz］	～5M	～33M	～300M	1～2G	～5G
性能［DMIPS］	～1	～10	～500	～3k	～10k
その他付加機構	−	高機能命令キャッシュ・メモリ（次世代でも継続）	遅延スロット	スーパスカラ，VLIW，投機実行・分岐予測，OoO実行	左記＋μOP実行，より高度な分岐予測など
設計の主な狙い	トランジスタの節約（使用効率）	1クロック当たりの性能向上	動作周波数の最大化	動作周波数を維持しつつ性能向上	設計複雑度低減とデータ移動最小化

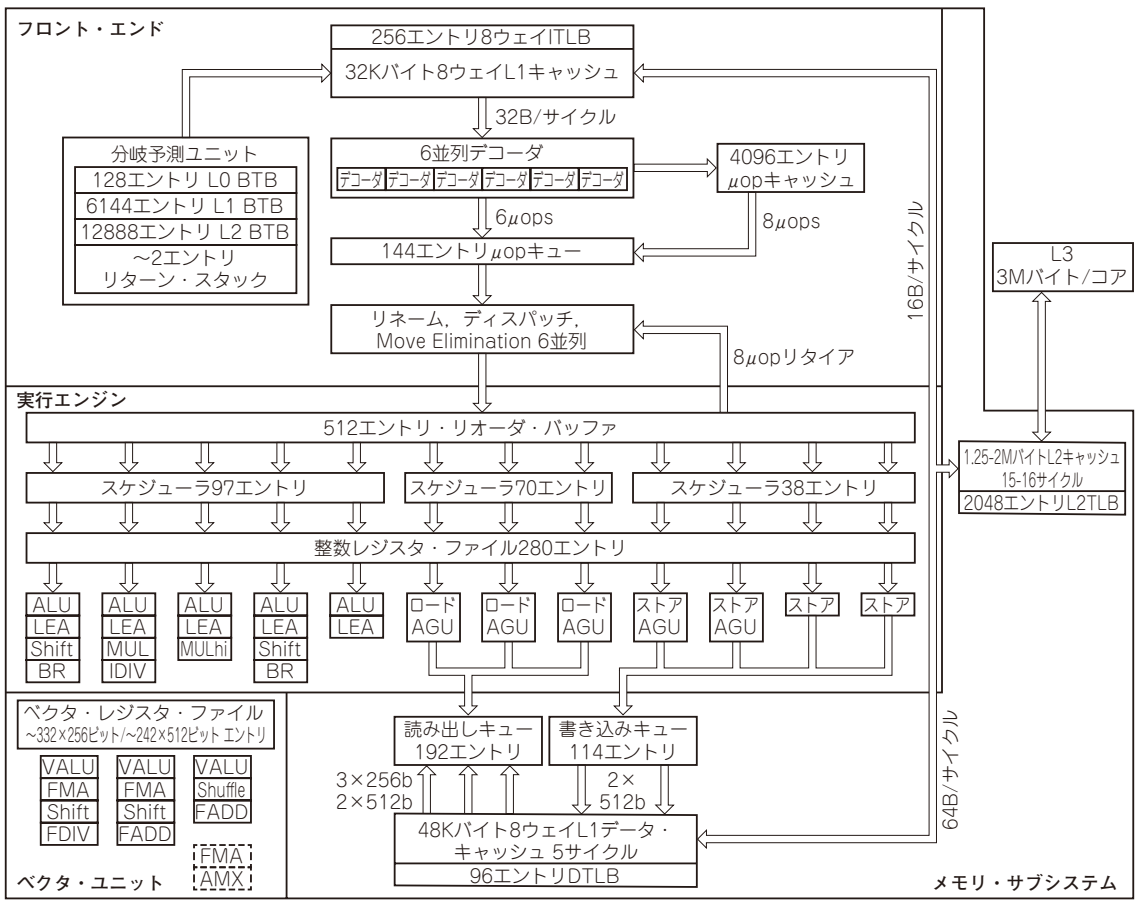

図2　近年のCPU（intel Golden Cove）**は高速化のために回路面積を割く構成となっている**
出典：Wikipedia（https://en.wikipedia.org/wiki/Golden_Cove）

結果としてCPUの命令数は増加し，数え方にもよりますが優に100種を超えるのが普通となりました（CISC ISA）．さらに外部メモリとCPUとの速度差の増大に対応するためにキャッシュ・メモリなども実装され，その容量も増大していきました．多数のレジスタや，10を超える演算器，それらをつなぐ複雑なデータ転送経路やそれを制御する回路など，非常に複雑な資源で構成されています（**図2**）．

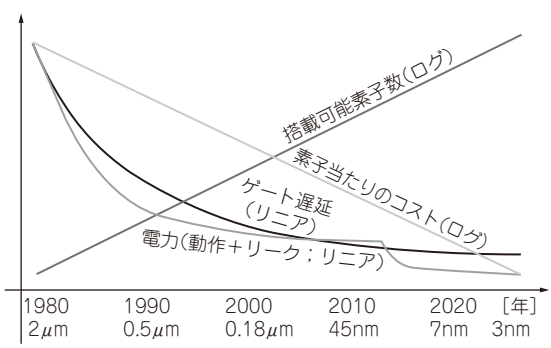

図3　製造プロセスが進歩するにつれ遅延時間は頭打ちになったが，トランジスタ密度は高まり続けている

▶現在…周波数向上は頭打ちに

性能向上のためにこのように変化しましたが，性能だけであれば必ずしも構成を大きく変更する必要はありません．動作周波数を上げれば，性能はリニアに向上し，かつソフトウェアの変更も不要です．

しかし，今やLSIの遅延時間を短くすることは容易でありません（**図3**）．一方で，トランジスタの密度は向上し続けており，それに応じてトランジスタ当たりのコストも低下し続けています．このLSI特性の変化を最大限に活用すべく，各社が開発競争をした結果，今日のCPUの姿になりました．LSIのプロセス・テクノロジとCPUの関係を**表1**に示します．

●ソフトウェアの進化もLSIを変えた

▶複雑化した制御部がボトルネックに

ソフトウェア技術の進化もLSIを大きく変化させました．増大した命令数に応じて命令デコーダやその制御回路などが複雑化しましたが，一方でキャッシュ・メモリの増大により，ある程度の処理ブロックであれ

ばメモリ内に収まるようになりました．すると結果として，複雑な命令デコーダや制御ロジックが，動作周波数のボトルネックとなり始めたのです．

命令数を減らすと，外部とのやり取りが多くなり，動作周波数が上がる代わりに，その待ち時間が増えるというジレンマに陥るのですが，コンパイラによる命令スケジューリングの最適化がこれを解決しました．

▶コンパイラによる最適化まで考慮した回路設計

ISAを単純な命令の組み合わせで定義し，代わりにレジスタ本数を増加させます．コンパイラが，命令と使用するレジスタおよび順序を最適化することで，多数のレジスタを動的にうまく使い，単純（理想的には命令デコーダが要らない）なISAであっても，外部とのやり取りを最小化することに成功しました．

これによってCPUの動作周波数は一気に2倍程度[注1]に向上し，さらにコンパイラ技術の進化とともに周波数当たりの性能（Instruction per Cycle；以降IPC）も向上していきました．

●命令実行順の入れ替えと分岐予測

2000年代前半までは，動作周波数とIPCの熾烈な向上競争が繰り広げられ，動作周波数をいっそう向上させるためにパイプライン段数は深くなり，アウト・オブ・オーダ実行が実装され，それに伴う分岐命令のペナルティを解消するためにさまざまな分岐予測機構が考案・実装されます[注2]．

●そして並列化競争へ

向上を続けた動作周波数とIPCですが，LSIのプロセス・テクノロジはいつまでもそれを許してはくれませんでした，物理的な小ささは配線抵抗を増大させ，アルミニウムから銅に配線材料を変えても，大きな改善は難しくなりました．

一方で，IPCを向上させるための仕組み，つまりキャッシュ・メモリによる命令/データ・アクセス帯域の向上や，深いパイプラインによってもたらされるデータ移動コストの相対的な低減は，新たな性能向上の可能性を生みました．それが並列化です（本稿ではCPU内での並列化だけを扱いマルチコアなどは対象としない）．

命令フェッチの帯域が十分にあり，かつパイプライン段数が深いと，大量の命令を事前にデコード（プリデコード）できます．シングル・イシュー（Single Issue；1クロック当たり1つの命令を処理）のCPUでも，その特性を利用してアウト・オブ・オーダ実行が可能となっていました．

この機構を利用して，スーパスカラ（Super Scaler）などのマルチイシュー（Multi-Issue；1クロック当たり2つ以上の命令を処理する機構）のCPUが一般的になります．さらに搭載可能なトランジスタ数の増大と，LSIの配線特性の変化（短くしないと速くならない）が，演算器の分割や複数化に拍車をかけ，最終的に**図2**のような複雑なマイクロアーキテクチャのCPUとなって今日に至ります．

扱うデータによって最適なCPU設計は異なる

●扱うデータの処理特性による分類

今日のCPUは内部での並列処理により性能を向上させていますが，どんな処理でも並列化できるわけではありません．複数の処理（命令）を並列に実行できるかどうかは，その処理間の依存関係に影響されるためです．更に，この依存関係を後回しにしたとしても，処理の並列化視点で見た特性（特徴）と並列化の手段とが合致していないとうまく処理できません．

図4に並列化視点での特性を，対称性と動的・静的という2軸で分類しました．実際には細かい特性も含めると到底図にできないほど複雑ですが，CPU視点で見るとこの2軸が最も影響の大きいものと考えて良いでしょう．

そしてこの2軸に，そこに向いている並列化手法を重ねてみると**図4**のようになります．それぞれの方式を説明するには誌面が足りないので，ここではポインタとなる用語の紹介にとどめます．しかし，明確なのは万能（全ての領域において最適）な並列化手法はないということです．

●処理特性と並列化の可否

処理の依存関係を考えます．**図5**（a）のシーケンス処理（シーケンサ）は，現在の状態と外部からの入力に応じて，次の状態と出力を決定するのが基本的な動作です．本質的に依存関係が強く，並列化が困難な処理と言えるでしょう．

一方で，**図5**（b）のデータフロー処理（データ・プロセッシング）は，固定的な処理の列を複数のデータが通過して行くイメージです．処理と処理の間で同じデータを扱う場合には依存関係がありますが，通常多くのデータが同じパスを通過するため，異なるデータ同士の依存関係は低いと考えられます．

現実の処理はここまで単純ではなく，特に近年は

注1：1990年代初頭でCISC ISA CPUが16MHz程度，RISC ISA CPUは33MHz程度で動作していました．

注2：OoOは，Out of Order Executionの略であり，パイプライン処理に都合が良いようにCPU内で命令の処理順を変更する機構です．分岐予測機構は分岐の方向を予測し，分岐条件の確定を待たずに予測方向の処理を実行する機構です．

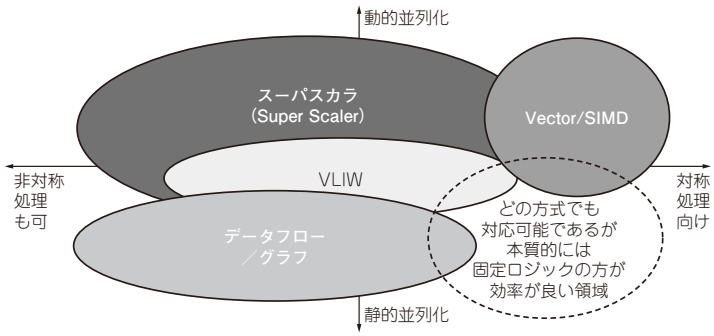

図4　並列化特性とCPU内での適切な並列化方式

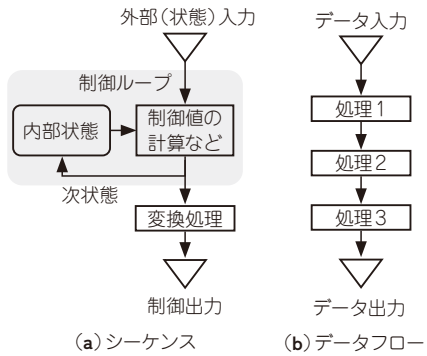

図5　特性の異なる処理のイメージ

これらが混ざった処理が大半でしょう．前者の代表格であるメカ制御でも，制御モデルの数値演算部分はデータフロー的であったりします．後者の代表格である画像処理（画像圧縮など）でも，画像の内容によって処理の分岐やデータフローの変化があったりします．

データの依存度を含めた処理特性と，並列化手法の有効性は密接に関係しています．多様な特性の処理を含んでいる場合，どの手法を用いたときでも，相性の良い部分は高速化されます．特にCPUのような，比較的デメリットが少ない[注3]方式の場合，全体として高速化されます．

しかし，処理が高速化されたからといって，それが最適な手法とは限らないことに注意が必要です．このことは，特に効率（電力やコスト）を重視する場合には，非常に重要です．

今日のプロセッサ・アーキと課題

●一般的なCPU設計の流れ

CPUの基本的な構成を論理的に描くと，**図6**のようになります．命令をフェッチして，デコードし，その結果に応じてレジスタを読み出し，演算などを実行し，結果をレジスタに書き込むというのが基本動作です．

CPUを設計する場合，動作周波数の狙い目を決める基準として技術的な唯一の正解はありませんが，メモリの速度か演算器（ALU）の速度をもとにする場合が多いと思います．これはこれらの資源が，内部構成上分割しにくいためです．なお，近年では搭載トランジスタ数の増大によりALU程度の資源であれば分割は難しくなく，メモリ周波数またはその倍数で動作周波数を決定することがほとんどでしょう．

動作周波数のターゲットが決まると，基本パイプラインのステージ数はおのずと決まります．**図6**の基本処理に必要な時間を動作周波数の逆数，つまりサイクル・タイムで割ればその数となります．

5ステージ・パイプラインを持つ初期のRISC CPUの例を**図7**に示します．この時代は，メモリ（キャッシュ）とALUの処理速度が近かったこともあり，バランスの良いマイクロアーキテクチャと言えるでしょう．

注3：CPUの場合，たとえ並列化できない処理だったとしても，並列化しない方式に対しての性能低下が少ない．

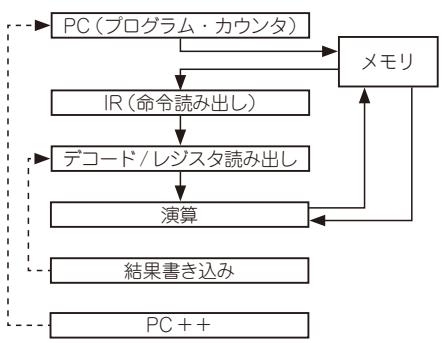

図6　CPUの基本的な構成

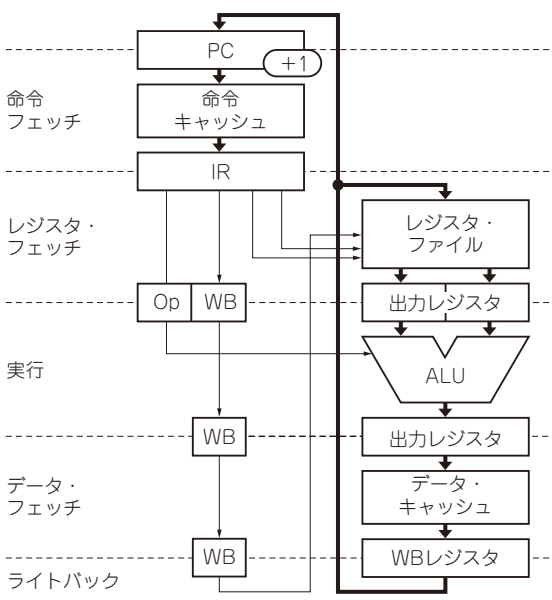

図7　シンプルなRISC CPUのパイプライン構成

●設計生産コストでは汎用的なCPUが有利だが…

実は**図7**の構成には無駄があります．命令の多くを占めるレジスタ-レジスタ間の演算命令は，命令フェッチ→レジスタ・フェッチ（デコード）→実行→書き込みの4ステージで十分です．しかし，データ（キャッシュ）メモリ・アクセスを伴う命令に合わせて1ステージ余計にデータが移動します．さらに悪いことに，この1ステージの移動中に後続の命令が結果を使用する場合，本来であれば結果をすぐに使えるのに，1ステージ待つという事態が発生します．対処する方法は幾つかありますが，いずれにしても効率をさらに下げることになります．

現代のCPUは，並列化のためにさらに資源が増加し続けており，**図8**に示すように本来の命令の処理に必要な資源の比率は低下の一途です．

ただし，比率が低いこと自体が即座に無駄が大きいという訳ではないことに注意が必要です．電力という面では，電力制御技術の進化もあり，回路を1つにまとめることが必ずしも良いとは言えません．また，回路規模≒コストという面では回路は小さい方が良いですが，1トランジスタ当たりのコストは年々低くなっており，本当に有意な差となるかは検証が必要です．特に商用CPUの場合，大量生産によるコスト・ダウン効果を考えると，極力同じCPUを共通で利用した方がコスト面でも有利になる場合がほとんどです．

LSIの，特に近年のLSIの怖さは，この共通化≒全体最適化の効果が，さまざまな個別最適化による効果よりも大きくなってきている部分にあると思います．結果として，動作周波数も並列性もそれほど重要でない用途であっても，複雑かつ大規模なCPUが使われるケースが増加しているのです．

●絶対性能と効率のトレードオフ

複雑化・大規模化が進むCPUですが，処理の視点に帰って考えます．**図9**に異なる特性の処理を，次のCPUそれぞれで実行する様子を示しました．

・5ステージ・シングル・イシュー
・15ステージ・6ウェイ・スーパスカラ

図9（a）は分岐が多くかつ分岐予測が当たりにくい例です．性能だけ見ると大差はないかもしれません．パイプラインのステージ数が増えると，通過するパイプライン・レジスタの数も増え，命令の実処理時間は長くなります．一方で，動作周波数の向上と命令並列化効果で，分岐予測ミスがない部分では大幅に命令スループットが向上します．

しかし電力については，いくら集積度が高くなり，トランジスタ当たりの電力が減少し，かつ電力制御の技術が進化したとしても，大量に投機実行され，かつ破棄された処理の分を取り戻すのは不可能です．

コストが見合う手段があれば，より処理に適したCPUを使えないか，検討する価値は十分にあると思います．そして，この差は今後のハイエンドCPUの方向性を考えると，益々増大すると思われます．

●最適CPU選択の可能性

▶ASICの場合

特定の処理に適したLSIとして，すぐに思いつくのはASICです．汎用を目指さないASICであれば，自由に適切なCPUコアを選択できます．近年では，RISC-Vのようなオープン・アーキテクチャもあり，以前より格段に処理に最適なCPUを開発または選択できる環境になってきています．実際に，RISC-Vが登場したころは，用途の多くがASICでした．

しかし，残念ながらまだまだ従来のCPUの域を出ていないというのが筆者の感想です．これは，おそらく技術的な問題ではなく，最適なCPUを使おうという意欲が強い人が，いわゆるコンピューティングの世界にのみ多いからではないでしょうか．

勝手な想像ですが，シーケンサのような処理を扱う開発者は，CPUを最適化してソフトウェアの実行効率を上げるより，論理回路を直接書いた方が早いと考えているのかもしれません．しかし，シーケンサのような制御にもプログラマビリティが求められてきている今日，一度論理回路と最適化したCPU（＋ソフトウェア）との定量的比較をしてみても良いのではないでしょうか．

▶FPGAなら最適なCPUを選びやすい

システムとしてハイエンドCPUが必要な処理があり，そのCPUでシーケンサなどの処理も行えるのであれば，追加でCPUを搭載する必要はないでしょう．

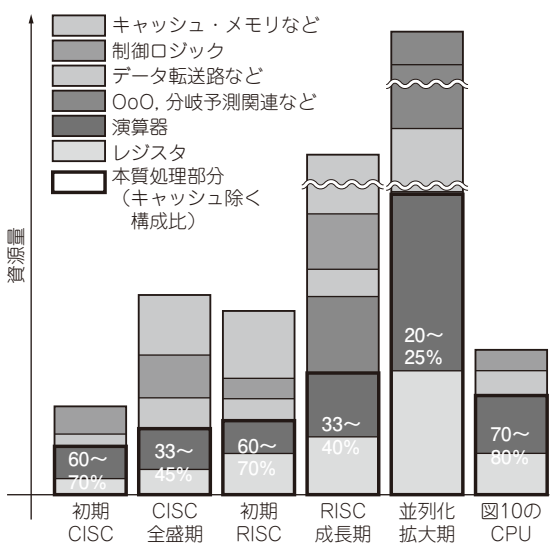

図8　CPUの資源配分のイメージ

しかし，もしCPUが必要ということだけで，ハード・マクロを持つFPGAデバイスを使っているのであれば，最適な構成を再検討する価値はありそうです．数年前に比べるとCPUを開発または入手するハードルは随分と低くなっており，参考になりそうなオープンかつフリーな実装も増加しています．

現代技術で作るコントローラ向けCPUの仕様を考察してみる

●マイコンなどのCPUに求められるもの

近年の高性能CPUはいったん忘れて，組み込み用のコントローラを考えてみます．組み込み機器での制御にもいろいろあり一意に決まりません．ここでは少し極端ですが，次の要件を仮置きして考えます．

- コード/データ・サイズは数百Kバイト程度（オンチップ・メモリに全て収容可能）
- まとまった処理セクションのコンテキスト・サイズ（データ/パラメータ数）は８～10程度
- データ/パラメータの必要精度・ダイナミック・レンジは10ビット・96dB（16ビット）程度
- 必要処理速度は100GOPS（MAC換算）程度

●命令セットによる差は少ない

以前は要件に最適なISAという視点も重要でした．しかし，現在ではLSIのプロセス・テクノロジが進歩した結果として，ISAによる差は無視できる程度になっています．ただし，次の点については要求をカバーできるかどうかを確認しておく必要があります．

- コンテキスト・サイズに対するレジスタ数
- データ長の精度とダイナミック・レンジ（ここでは固定小数点数を想定）
- 主要な（性能に影響する）演算などの命令処理種別

ここでは，上記を満たした上でソフトウェア開発環境も含めた入手性が良い点を評価して，RISC-V（RV32IM）[1]を選択します．

●マイクロ・アーキテクチャ

次の点を考慮し，3/4可変ステージ・シングル・イシュー構成とします．

- 100GOPSのスループットをRISC ISAで実現するために3倍の300MHz動作をターゲット
- 上記とオンチップ・メモリにコード/データを全て収容可能なことから，メモリ・アクセスに1ステージ割り当て
- 300MHz動作であれば，ALU/MACは1ステージで十分なので，最低限の1ステージを割り当て（将来浮動小数点数演算が必要な場合は2ステージを割り当てることを想定）
- 動作電力低減を優先しALU，MAC，DIVは個別実装（将来浮動小数点数演算をサポートする際にも容易）
- 上記＋分岐高速化のために，分岐先アドレス計算器も個別実装
- 演算器出力とデータ・メモリ出力は後続命令のインタロックを最小化するために実行ステージ入力へバイパス経路を実装

可変ステージ構成とすることで，LD命令のデータ・

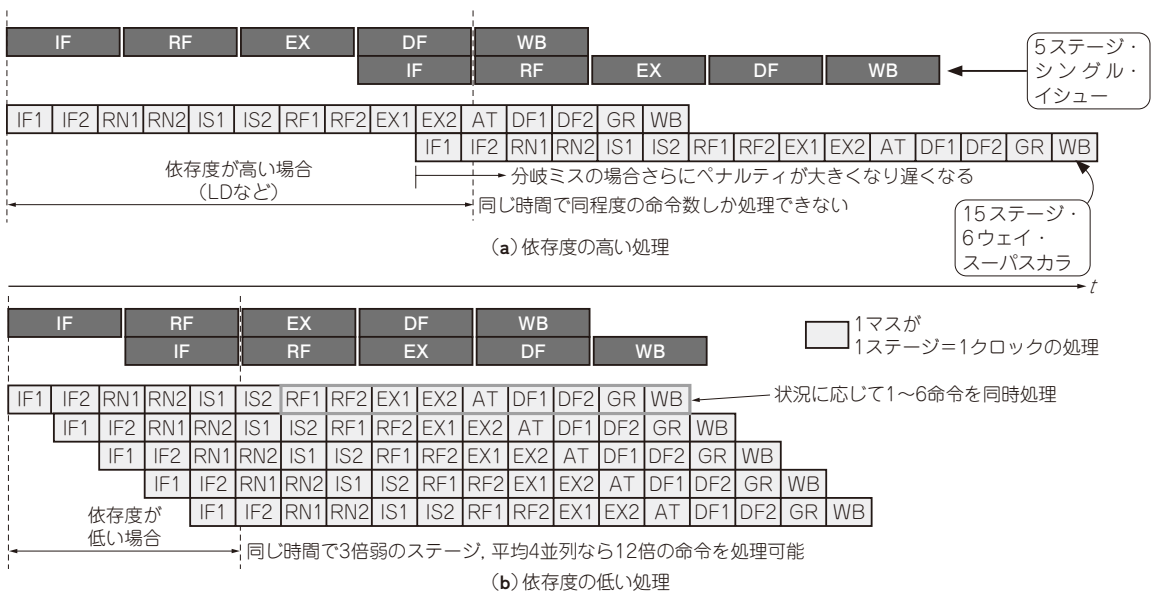

図9　処理の特性によって最適なCPUアーキテクチャが異なる

IF：Instruction Fetch；命令フェッチ，RN：(Register) ReNaming；(レジスタ) リネーミング，IS：(Instruction) ISsue；命令発行，RF：Register Fetch；レジスタ・フェッチ，EX：EXecution；実行，AT：Address Translation；アドレス変換，DF：Data Fetch；データ・フェッチ，GR：GRaduation；命令完了，WB：Write Back；ライトバック

メモリからの出力と，後続の演算命令の演算結果出力のレジスタ書き込みタイミングが競合します．

しかし，データ・メモリ出力は常にバイパス可能で，かつ次のLD命令の演算ステージでは必ずレジスタ書き込みがないため，そのタイミングまでパイプライン・レジスタ（LDバッファ）に保持して書き込む構成とします（**図10**）．

●命令の種類ごとのステージ数

この構成での代表的な命令処理を**図10**に示します．

▶演算命令

通常最も実行頻度の高い演算命令は最短の3ステージで完了し，使用する資源も最小です［**図10**（**a**）］．

▶LD/ST命令

この構成では最長となる5ステージを要します［**図10**（**b**）］．なお，浮動小数点数演算をサポートする場合には，浮動小数点数演算器（FPU）を用いて，このパイプライン動作と同じように処理できます．

▶分岐命令

制御CPUで性能の要となる分岐命令［**図10**（**c**）］は，結果の書き込みが不要なので，実質2ステージで完了し…たいところですが，RISC-Vの条件分岐命令はレジスタの比較が必要です．条件が確定する実行ステージを待つため3ステージ必要です．再フェッチも必要なので実行時間も3クロックを要します．ここはフラグを持つISAであれば1クロック縮められるので残念なところです．ただし，フラグの生成に余計な命令を使わないといけない場合もあるため，常にフラグを持つISAが優秀という訳ではありません．

●効率に関するトレードオフ

今回採用していない機構も含め，高速化に用いる機構が持つ要件に対するトレードオフを**表2**にまとめます．

資源の表現の詳細度がかなり異なりますが，このCPUは現在の主要なCPU（**図2**）と比べて，かなりシンプルな構成です．特に，アウト・オブ・オーダ実行と分岐予測および投機実行に必要な資源は非常に大きく，扱うデータ量とデータ・サイズが相対的に小さい制御処理にとっては，無駄の大きな構成です．

図8では，**表2**に挙げた機構の中で，要件に照らし合わせて必要な部分だけを太枠で囲ってあります．現代のCPUがいくら全体最適を追った結果とはいえ，処理要件によっては無駄が多いことが分かります．

実はこのCPUですが，ISAが異なるので細部は異

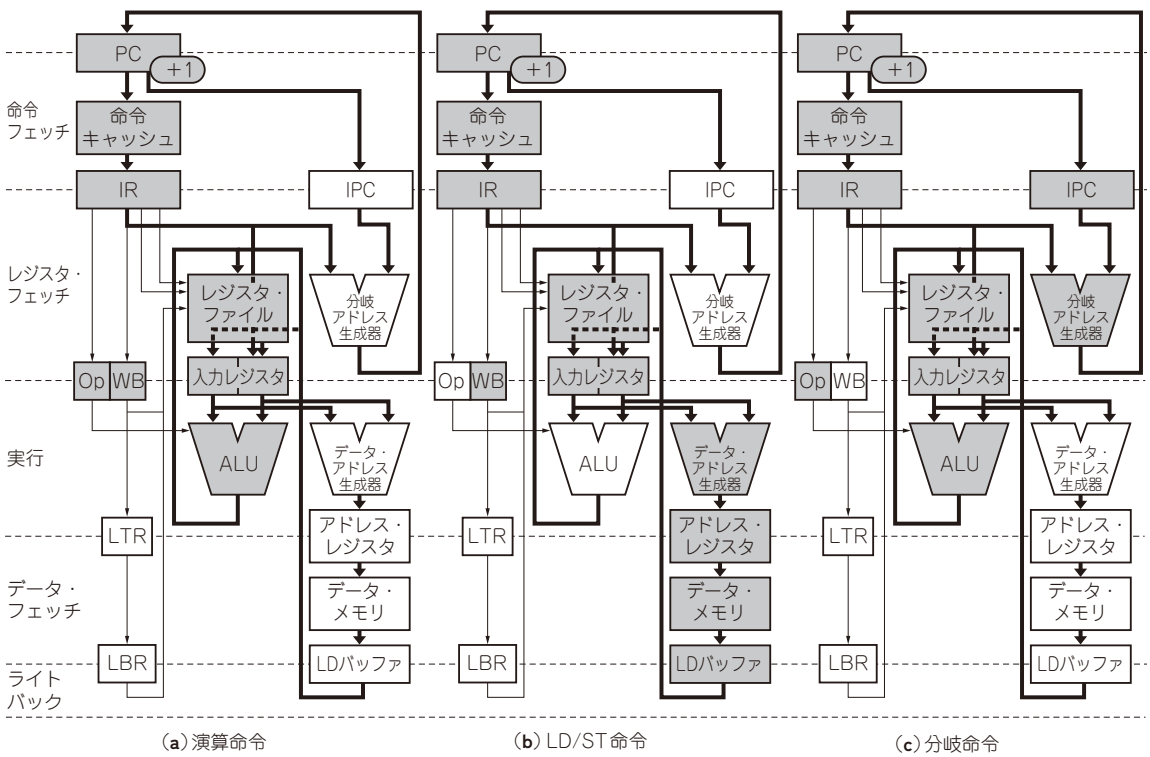

図10　制御用に構成を考えたCPUでの命令の処理例

IR：Instruction Register；命令レジスタ，IPC：IRと同期したPC，Op：Opecode；オペオード（演算種類など）を保持するレジスタ，
WB：Write Back Register Address；ライトバックするレジスタの番号を保持するレジスタ，LTR：LD結果をライトバックするレジスタの番号を一時的に保持するレジスタ，LBR：LDバッファにあるデータがライトバックされるレジスタの番号を保持するレジスタ

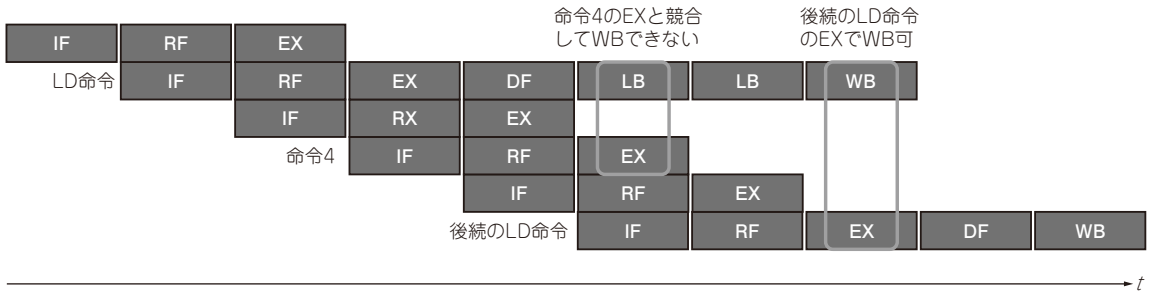

図11　LDバッファを持つ構成での書き込み処理

なりますが，**図11**の方式も含め筆者が過去に設計し量産されたCPUとほぼ同じ構成です．どのCPUか分かるでしょうか．当時のLSIのプロセス・テクノロジでは300MHzで動かすのは厳しく，せいぜい150MHz程度だったはずですが，制御向けのCPUとしては，それなりに高効率だったと思います．マイクロアーキテクチャとしては，残っていれば今でも通用したかもしれません．しかし，商用CPUとして見ると，たとえ短期的にでも性能競争に勝てなくなれば消えて行ってしまうというのが少し寂しいところです．

このクラスのCPUが適している用途は数多くある

表2　処理要件とCPUの各種機構のトレードオフ

高速化手段	制御処理中心	データ処理中心
動作周波数向上	◎	◎
多段パイプライン	× 一般に依存性高く性能低下する．ただし，動作周波数向上とのトレードオフ	○ 多くの場合，分岐予測などとの併用で性能低下をカバーできる
分岐予測・投機実行	× 当たらない！	○
アウト・オブ・オーダ実行	△ 通常分岐間隔が短く効果が低い	○ 並列化にはほぼ必須
並列実行（スーパスカラ，VLIWなど）	△ 通常分岐間隔が短く効果が低い	◎
複合命令追加	△ bit操作やbit test &Branchなど有効なものもある	× 制御複雑化によるデメリットの方が大きい場合が多い
キャッシュ・メモリ	△〜○ 必要メモリ・サイズに依存	◎

と思います．300MHzのCPUでよいのに，相性の良くない2GHzのCPUを使っているケースも散見されます．汎用もしくは標準品のCPUとしては消えてしまっても，別な形で開発者の手に届くようになり，そのCPUを使った開発がより効率的かつ競争力を持ったものになることを期待しています．

シンプルに1ステージで実装

●単純なシーケンサ実装の選択肢

高速かつ高精度な制御処理の場合，適応制御やノンリニアな補正・フィードバックなどを行うことが多く，100GOPS程度の演算処理性能が必要な場面も多くあります．一方で単純なシーケンス処理だけが必要な場面もあり，より単純なシーケンサを実現する手段として，次のような選択肢が考えられるでしょう．

- 論理回路でシーケンサを実現する
- 究極に簡単かつ効率の良いCPUを使う
- 一般的なCPUを使う

これに加え，処理特性[注4]や実装方法（固定論理かプログラマブルか）といった条件によって，適した選択肢が変わります（**表3**）．ある程度状態数や遷移が複雑で，かつその遷移が多くの外部情報に依存する場合は，CPUという選択肢も視野に入ってくるでしょう．

●演算回路の最適化こそCPUの強み

▶回路共有と最適化

表3を細かく見ると，演算処理の分布がCPU採用のカギになります．一般に，シーケンサを独自の論理

注4：状態数の多さや遷移の複雑さ，外部とのデータのやり取りの多さなど．

表3　シーケンサの処理特性と実現手段

処理特性 ＼ 実現手段	単純なシーケンス	複雑なシーケンス	
		固定的処理（時代が変わっても）	変化の大きい処理（進化途中など）
AISCなど（固定ロジック）	固定ロジック ただし，CPUがあればCPU	固定ロジックまたは＋CPU	CPUなど プロセッサ＋ソフトウェア
FPGAなど（プログラマブル）	固定ロジック ただし，CPUがあればCPU	固定ロジック	CPUなど または固定ロジック

回路で構成する場合，必要な演算処理をどのような演算器配置で実現し，それらをどのように共有するかを決定することは容易ではありません．最近の論理合成ツールが持つ自動リソースシェア機能はかなり賢くなっています．しかし，それは静的，つまり回路構成から推定可能な情報のみに基く最適化です．動的な情報を使った最適化はできません．つまり実際の動作の条件では同時動作しないものの，静的にはそれが判断できない部分に対しては，設計者が明示的に適切な回路を構成するか，個別にそれを指示する必要があります．

▶処理に占める演算比率が高いならCPU

CPUは演算回路を最大限に共有します．演算処理を含まない純粋なシーケンサ部分の効率で論理回路が勝っても，演算の頻度または比率が増加するにつれて，全体効率ではCPUが勝つ可能性があります．

どのような条件であればCPUが効率で勝るかは，さまざまな条件に依存します．条件を仮置きするしかないのですが，ここでは例として，少しCPUに悲観的な次の条件で考えてみます．

- CPUは命令フェッチが必要であり，基本的な効率は1/2である
- 外部とのデータのやり取りおよび演算については，CPUの方がレジスタなどにデータを集約でき，データ移動距離が短いため3倍効率が良い（命令フェッチ込みで1.5倍良い）
- 状態遷移処理は，メモリやレジスタ上のデータおよびその演算で表現する（直接論理でない）CPUが3倍効率が悪い（命令フェッチ込みで6倍悪い）

この前提において，シーケンス処理中の演算割合をxとすると，CPUの論理回路に対する相対効率は次の通りです．

$$1/\{1/1.5x+6(1-x)\}$$

計算すると，演算の比率が94%程度になるとCPUの方が効率が良いことになります（**図12**）．実際には，命令フェッチのオーバヘッドはここまで大きくないと思いますし，シーケンス処理の効率も改善可能な面があります．またCPUには，コンパイラによってシーケンスや演算をオフラインで（つまり実行時に時間や電力を使わずに）最適化できるという武器もあるので，差は小さくなると思います．

●50MHz程度ならパイプライニングも不要

CPUをもっと改善するヒントは**図6**と**図10**に隠れています．**図10**のマイクロアーキテクチャでは制御系としてはやや高めの100GOPSをターゲットにしました．しかし，シーケンサには1μs程度の制御周期で十分な用途も多数存在します．

仮に1シーケンスに50命令の処理が必要だったとしても50MHzあれば十分です．その場合，パイプラインにする必要もなく，純粋に**図6**の構成を全て1クロックで処理する1ステージCPUで十分です．これにより，パイプライン・レジスタやそこに供給されるクロックの経路，パイプラインの多段化に伴うバイパス回路およびその制御回路などを削減でき，より純粋な演算処理効率に特化したCPUとなります．

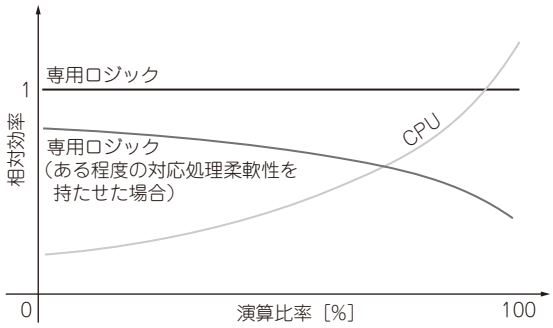

図12　CPUの資源配分のイメージ

この構成は初期のCPUに近いのですが，大きく異なる点はLSIの制約にあります．当時はせいぜい数万トランジスタの集積と数MHzでの動作が精いっぱいで，性能向上のためにさまざまな付加機構が追加されていました．しかし，現在ではLSIの進化によって，CPUにとっては実質無制限にトランジスタを使いつつ数十MHzで動作可能です．古くて新しい基本に立ち返った実装が可能になったと言えるのではないでしょうか．

＊　＊　＊

本稿で紹介した内容は，CPU全体の複雑度からすればごく一部であり，実際には数多くのトレードオフと戦いながら設計を進めます．全てのトレードオフの判断を経験できなかったとしても，主要なトレードオフをCPU設計の視点で経験することは，CPUを開発する上ではもちろん，CPUを選定したり使ったりする上でも有用なはずです．

近年CPUを作ることは容易になってきています．RISC-Vのようなオープンな ISAを使えばもちろんですが，独自ISAのCPUでさえ，LLVMのようなコンパイラ・フレームワークが充実してきたことで個人でも作って動かすことができます．

品質やサポートを求めると簡単ではありませんが，CPUの設計を経験するという目的であれば，誰でも手の届くところまで来ています．皆さんも，自分なりの理想のCPUを作ってみてはいかがでしょうか．

◆ 参考文献 ◆

(1) The RISC-V Instruction Set Manual Volume I.

すぎもと・ひでき