（1） CPU の高速化

これまでの紹介では、CPU の機能、内部のしくみの概略についてご説明してきました。今回は、実際の CPU の開発者が求めているもの、いかに速い CPU を開発するか。そこに焦点を当ててみたいと思います。どのような機械でも、その開発者は他より優れたものを作りたいという本能があります。CPU 技術者の場合は高速化がその最大の関心事となります。「2 番じゃダメなんですか？」という有名な発言がありましたが、「2 番じゃダメなんです！！」というのが、技術者の回答ですね。

CPU の高速化を実現するためには、どのようなポイントがあるか、CPU のしくみをおさらいしながら、調べてみましょう。

CPU の機能はマイクロプログラムを実行することによって実現されているということをご説明しました。どのような処理をしているかというと、

「レジスターの内容を読んで、演算して、レジスターに書く。」

この処理の中では、「演算をする」というところが実は一番回路も大きく、時間もかかります。それでは、ここを速くしましょう！

次には、

「メモリーのデータとレジスターを読んで、演算をして、レジスターに書く。」

この場合、「演算をする」よりも、「メモリーのデータを読む」という処理のほうが時間がかかります。それでは、ここを速くしましょう！

一人でやるより、大勢でやったほうが速く処理できる。というのは、どんな仕事でもいえることだと思いますが、CPU についても当然重要な高速の考え方です。

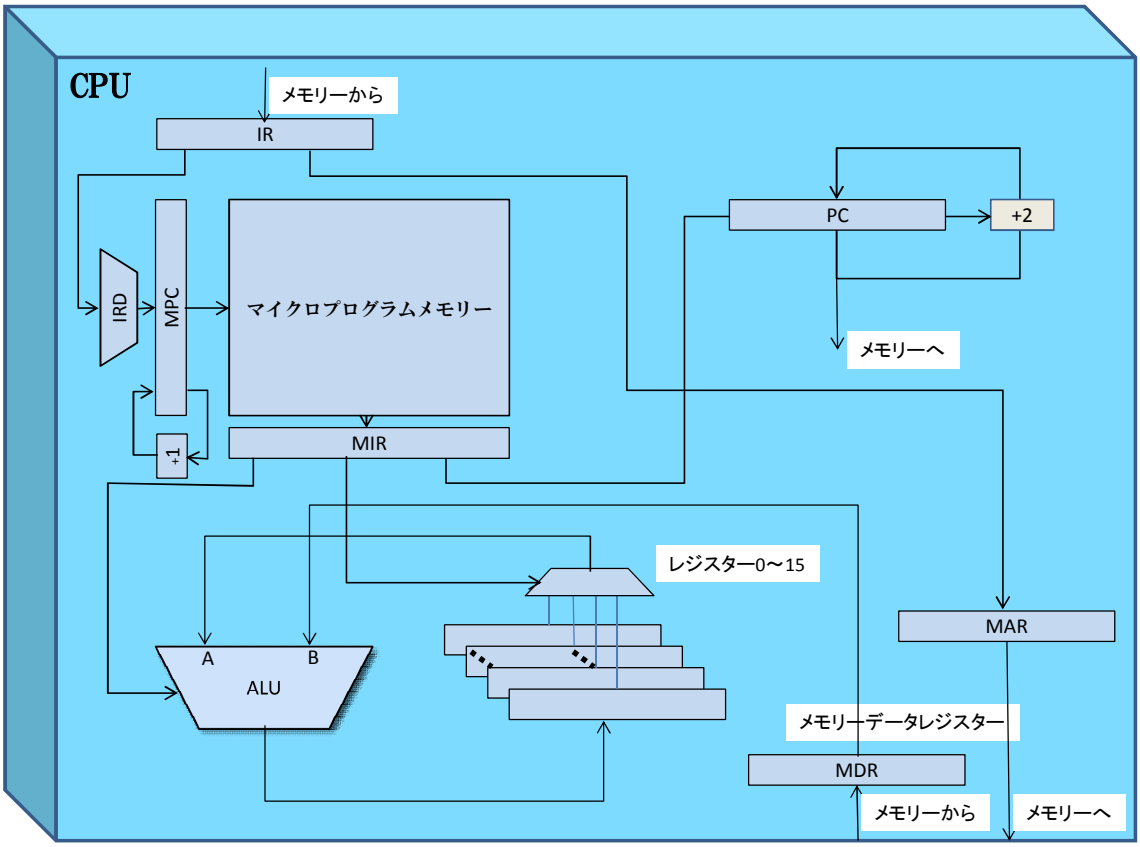
同じ機能を実現するにしても、やはり工夫次第で処理時間が大幅に改善されるということはよくあることです。CPU でもこれを検討しましょう。

ということで、「マシンサイクルの短縮」、「メモリーの階層化」、「並列化」、「アルゴリズムの工夫」の 4 点について、それぞれどのような工夫がなされているかをご紹介したいと思います。

（2） マシンサイクルの短縮

下に、また例の CPU 内部のブロック図を示します。ブロック図という言葉は初めて出てきましたが、レジスターや演算装置など、機能の単位をひと固まり(箱)として表現した図で、要は下のような図のことです。回路図になると、それぞれの機能単位がさらに細かい部品レベルの表現になります。

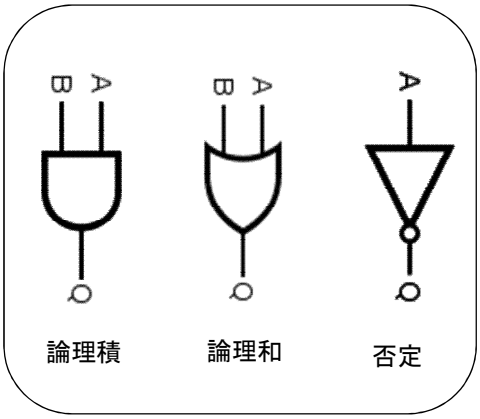
さて、このブロック図を見ながら、マシンサイクルについて、説明します。マシンサイクルというのは、記憶装置（レジスター）から読み出し、演算処理し、レジスターに書き込む一連の処理が含まれます。そのマシンサイクルの変わり目には「クロック」という信号が出されます。２GHｚなど、クロック周波数などがプロセッサの性能でよく使われますが、そのクロック信号というのが、これです。レジスターというのは、クロック信号が入ることによって、その入力データを取り込みます。つまり、クロック信号はレジスターの内容を変更するタイミングを与える意味もあり、クロック信号から次のクロック信号までを１マシンサイクルと呼び、この長さがいかに短いかが性能の指標になるわけです。



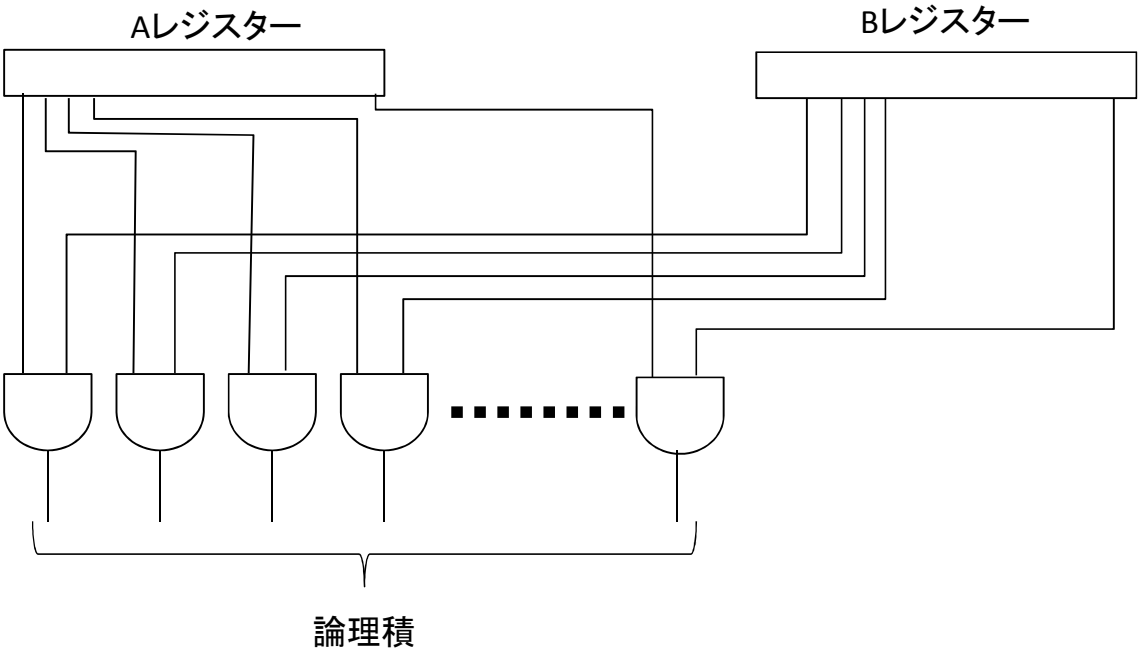
さて、レジスターを読んで、演算処理して、レジスターに書く。という処理の中で、やはり一番時間を要するのが、演算処理です。

演算を処理する算術論理演算装置（ALU）というのはコンピュータの中のコンピュータといってもよいくらい、中心的な存在です。演算処理は、加減算、論理積、論理和、否定、シフトなどの演算処理をしますが、これらの処理は、論理積、論理和、否定の３種の論理回路（下図）の組み合わせで実現されるデジタル回路で構成されます。論理積というのは、入力ＡとＢが両方とも「1」のときのみ、出力Ｑが「1」になり、それ以外は出力Ｑは「0」です。論理和は入力ＡとＢのどちらかが「1」のときに、出力Ｑが「1」になり、両方とも

「0」のときのみ出力Qが「0」になります。否定は、入力Aが「0」のとき出力Qは「1」、入力Aが「1」のとき出力Qは「0」になります。つまり反転です。そのような3種類の回路（ゲート）の組み合わせで実現されます。



論理演算（論理積、論理和）は、データのそれぞれのビットごとに論理積ゲートや論理和ゲートが一つずつあればいいだけですので、非常に簡単な回路で実現できます（例えば論理積は下図参照）。



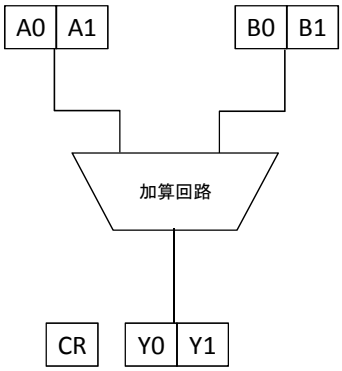
一方、我々人間が計算するときは、加減算というのが、計算の基本ですから、一番簡単なように思えますが、実は、デジタルの世界では、加減算というのはなかなか厄介な処理なのです。

なぜ、やっかいなのか？

それは、下の桁からの桁上がりを考慮した計算をしなければならないからです。

2 ビット同士の加算を考えてみた場合、上の桁は下の桁が両方 1 の場合は桁上がりがあるため、上の桁同士だけでは演算ができません。

2 ビット同士の加算回路を作って、桁上がりをどのように処理しているのかを調べてみたいと思いますが、いきなりでは理解しづらいので、まずは入力の値と出力の値の一覧表を作ってみましょう。入力は A と B でそれぞれ 2 ビットですから A0、A1、B0、B1 とし、加算回路に入り、出力 Y0、Y1 の 2 ビットと桁上がり CR を出力するものとします。



出力

この図を見ながら、出力の Y0、Y1、CR がどのようなときに 1 になるか考えてみましょう。下の表が、入力 A、B と出力 CR、Y の関係です。このような表を真理値表と呼び、論理回路を設計するときの元となる表です。表の見方は、例えば、A0、A1 が 01 で、B0、B1 が 10 のとき、つまり、A0、A1 が 01 の行を見て、B0、B1 が 10 となっている列のセルを見てみると、011 となっています。これは、CR=0、Y0=1、Y1=1 が出力されるという意味です。このようにして、出力のそれぞれが 1 になるときの入力値がどうなっているかを調べます。

入力A0、A1とB0、B1 と 出力 CR、Y0、Y1					
		B0、B1			
		00	01	10	11
A0、A1	00	000	001	010	011
	01	001	010	011	100
	10	010	011	100	101
	11	011	100	101	110

まず、Y1 についてみてみましょう。下に Y1 の真理値表を示します。

論理回路を設計する場合は、真理値表の出力が 1 になる条件を論理回路にする方法がとられます。まず、B1 が 1 の列は B0 が 0 でも 1 でも同じです。さらにその条件で A 入力を見てみると、A0 が 0 でも 1 でも同じです。つまり、A1=0 AND B1=1 という条件のとき Y1 は 1 になります。さらに B1 が 0 の列でも、B0、A0 が 0 でも 1 でも出力は同じなので、A1=1 AND B1=0 という条件でも Y1 が 1 になっていることがわかります。つまり、(A1=0 AND B1=1)

OR (A1=1 AND B1=0) という論理式が出力 Y1 の条件です。この真理値表を用いる設計方法は出力が 1 になる条件をすべて OR で並べればいいので、単純なのですが、ここで設計しようとしている回路は、加算という処理であることがわかっており、その意味を考えながら論理式を考えてみると、回路の意味が論理的に納得できるものになることが多いようです。それではやってみましょう。Y1 は下の桁なので、桁上がりなどは考えなくていいので、Y1 は A1 と B1 の値のみによって決まります。A1、B1 両方 0 のときは 0、両方 1 のときも 0、どちらかが 1 でもう一方が 0 のときに 1 となっていることは加算の意味がわかっていれば、当然ですね。

入力A0、A1とB0、B1 と 出力 Y1

		B0、B1			
		00	01	10	11
A0、A1	00	0	1	0	1
	01	1	0	1	0
	10	0	1	0	1
	11	1	0	1	0

次に Y0 です。今度は真理値表からではなく、加算の意味を考えながら設計をします。Y0 は上の桁なので、下の桁からの桁上がりを考慮しなければならないことがわかります。下からの桁上がりは A1 と B1 両方とも 1 のときに桁上がりがあります。つまり A1 と B1 のどちらかが 0 ときは、A0 と B0 のどちらかが 1 でもう一方が 0 のときに 1、A1 と B1 両方とも 1 のときは、A0 と B0 両方とも 0 か両方とも 1 のときに 1 となります。

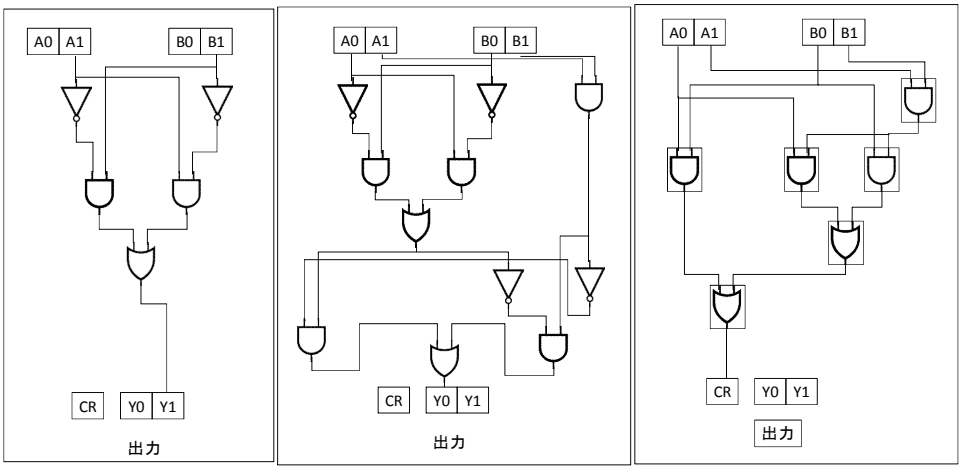
次に CR です。まず A0 と B0 両方とも 1 のときは 1 になります。A0 と B0 どちらかだけが 1 の場合で、下からの桁上がりがあるとき、つまり A1 と B1 両方とも 1 のときにも 1 となります。

さて、以上を回路にしてみますと、それぞれの出力ビットの回路は、下の 3 つの図のようになります。左から Y1、Y0、CR 出力の演算回路を示しています。すべて論理積、論理和、否定の 3 種類のゲートで構成されています。ちなみに、Y1 を出力する論理回路は A1 と B1 の排他的論理和となっています。逆に排他的論理和を論理積、論理和、否定の 3 種の回路で実現するとこのようになる、ということです。排他的論理和を使うとこの回路は単純に表せますね。でも回路の素子数としてはどちらでも同じです。

さて、高速化という観点に話題を戻してみます。

Y1 と Y0 の回路を比べてみますと、Y0 の方が複雑になっているのがわかりますし、入力から出力までのゲート段数、つまりいくつのゲート（論理積、論理和または否定）を通っているかを数えると、Y1 は 3 段なのに対し、Y0 は 6 段となっています。この段数こそが、

処理時間に直接関係してくる数字で、この段数をいかに少なくするかが、マシンサイクルの短縮にとっては重要になってきます。



この 2 ビットが 32 ビットになったときを想像してみてください。1 番上のビットの演算結果は、下の 31 ビットの演算結果をすべて調べてからはじめて演算できる、という具合です。そう考えると、回路の量を減らすためには、データ長が 32 ビットなら、32 マシンサイクル必要になるのか？というような疑問も出てきます。しかし、そこまでやっていたのでは、高速化は望めません。そこで考案されたのが、キャリールックアヘッドという仕組みです。つまり、それぞれの桁について、その下位の桁の値を入力とした、桁上がりがあるかどうかを調べる論理回路を作り、その出力を入れてその桁の演算をするという仕組みにして、桁上がりを処理するのです。2 ビットの加算でも A1 と B1 の論理積をとる回路がありましたが、簡単ですがこれも一種のキャリールックアヘッド回路ですね。

なんだ、簡単じゃないか！とお思いでしょうが、現実はなかなか厳しく、桁上がりを調べる回路はデータ長が長くなれば長くなるだけ膨大となり、32 ビット幅の演算装置というのは、昔の LSI の集積技術では、一つのチップには入りきらないという問題があり、データ幅を 16 ビットにするなどの限界があった時期もありました。現在は 64 ビット ALU も出ておりますので、集積技術の進歩に驚くばかりです。つまり、半導体の集積技術の向上は、回路を小さくし、製品を小型化する効果はもちろんですが、演算の高速化にきわめて大きな意義があったのです。

（3） 記憶装置の階層化

マシンサイクルの定義では、レジスターからレジスターまでの時間が 1 マシンサイクルでしたが、レジスターとメモリー（主記憶装置、以降単にメモリー）の間の演算では、1 マシンサイクル内では収まりません。それは、メモリーからデータを読み出す時間が大きいからです。実際、メモリーからデータを読み出す時間は、レジスターから読む時間に比

べ、数倍から十倍近くかかる場合もあります。命令の実行時間とは直接関係はありませんが、ハードディスクとメモリーのアクセス時間も大きな差があります。1ケタ～2ケタも処理時間が違うこともあるようですので、パソコンの高速化を手っ取り早く実現しようと思ったら、メモリーの増設が一番効果が大きいようです。話がそれたついでに、さらに余談ですが・・・企業活動でも、自部門内で処理できる範囲であれば、非常に高速に処理が終わるのに、他部門との調整が必要な場合は、数倍、他社との調整が必要だとさらにその数倍、それが海外になるとさらに十倍などと、時間がかかるのに似ていますね。余談でした。

メモリーとレジスター間の演算を何とか1マシンサイクル内で処理するためにはどうすればよいか？

CPU内にメモリーデータの写しでも持っておけばいいのでは？そうです。CPU内のメモリーであればレジスターと同等ですから､1マシンサイクル内で処理することができます。そんな考えで考案されたのが、キャッシュメモリーという記憶装置です。Cache Memoryと書きますが、このキャッシュというのは隠すという意味があるそうです。つまり、実際のメモリーのアクセス時間が遅いことを覆い隠して高速にアクセスできるようにするメモリーとでも言ったらよいでしょうか。

キャッシュメモリーは、一度CPUからメモリーへのアクセスがあるとそのデータおよびそれを含むブロック(ある程度の塊、メモリーアクセスの実現方式により決まる)を同時に読み、キャッシュに登録します。次回以降同じアドレスや、そのブロック内のアドレスをアクセスするとキャッシュからのアクセスとなるため、高速になる。という具合です。

以上、記憶装置については、

一番高速なチップ内の記憶装置

↓

チップが搭載されているプリント基板上の記憶装置

↓

別のプリント基板上にある主記憶装置

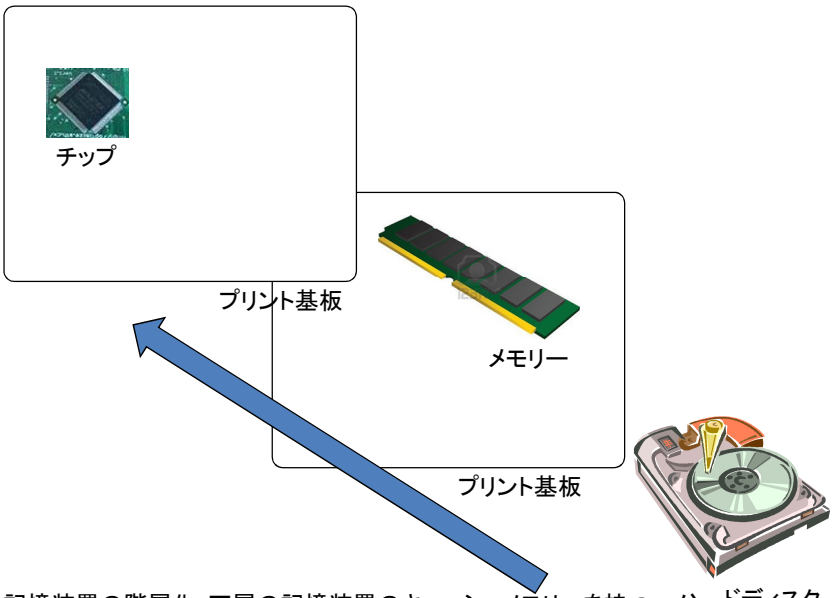
↓

CPUの外にあるハードディスク装置

↓

・・・

という具合に、徐々に低速になる下位の記憶装置のキャッシュメモリーを持つことによって高速化を維持しています。（下図参照）



記憶装置の階層化：下層の記憶装置のキャッシュメモリーを持つ

今回は、「マシンサイクルの短縮」、「メモリーの階層化」という 2 点について、コンピュータを高速化するにあたってのキーとなっていることをご理解いただきました。マシンサイクルの短縮には ALU のビット幅を大きくするためキャリールックアヘッドとそれを実現するための集積技術の高度化が重要でした。メモリーの階層化はキャッシュメモリーという方式の導入が鍵でした。次回も高速化の工夫として、並列化、アルゴリズムの工夫を紹介します。