計算機システムⅠ・第14回

2022年7月28日 修正版

今回の内容

14.1 クロック周波数の上限

前回紹介した単純な4 bit CPUは、クロック信号の1周期[1]ごとに、1つの命令を実行することができます。クロック信号の周期を短くする、つまりクロック信号の周波数を高くすれば、このCPUは、より高速に機械語命令を実行していくことができるわけですが、それには限界があります。たとえばCPY A, B命令のように、Aレジスタの値を変える命令が実行される状況を例として考えてみると、次のような順序で信号が伝わって行きます。

1. クロック信号 (CLK) が立ち上がる[2]。
2. PC (プログラムカウンタ) の出力 $Q_5\,Q_4\,Q_3\,\ldots\,Q_0$ が1だけ増える。
3. これにより、プログラムメモリのアドレス入力 $A_5\,A_4\,A_3\,\ldots\,A_0$ が変化し、この命令 (CPY A, B) のビットパターンが $D_{o7}\,D_{o6}\,D_{o5}\,\ldots\,D_{o0}$ に出力される。
4. これが、デコーダの $I_7\,I_6\,I_5\,\ldots\,I_0$ に入力され、デコーダの出力 S_1^A, S_0^A が変化する。
5. これが、Aレジスタの入力セレクタの S_1^A, S_0^A に入力され、出力 $Y_7\,Y_6\,Y_5\,\ldots\,Y_0$ が、Bレジスタの (出力 Q の) 値に変化する。

ここまでの過程が完了していないと、クロック信号の次の立ち上がりでAレジスタの値を正しく設定することはできません。ところが、この過程では、多くの論理素子を経由して信号が伝わりますので、どうしても一定の時間が必要となります。この時間がクロック信号の周期の下限、つまり周波数の上限となります。

14.2 パイプライン処理

CPUにより高速に命令を実行させるために、上の1から5までの過程をいくつかの段階に分割し、それぞれの段階をクロック信号の1周期で実行するという手法があります。たとえば、クロック信号の立ち上がり以降の処理を、次の3つの段階に分けて、それぞれを、クロック信号の1周期の時間で処理します。

STAGE 1. PC (プログラムカウンタ) の出力 $Q_5\,Q_4\,Q_3\,\ldots\,Q_0$ が変化して、プログラムメモリの出力 $D_{o7}\,D_{o6}\,D_{o5}\,\ldots\,D_{o0}$ が定まる。

STAGE 2. デコーダの出力 LD^{PC}, S_1^A, S_0^A, S_1^B, S_0^B, S^C, S^O, $\overline{A}/S$ が定まる。

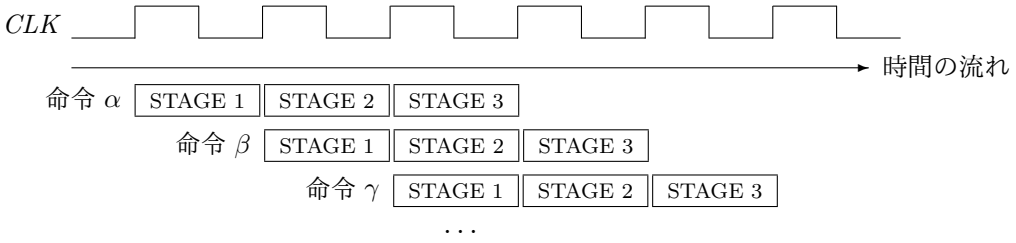
STAGE 3. 各レジスタ (A, B, Out, CF, PC) の入力 D が定まる。

---

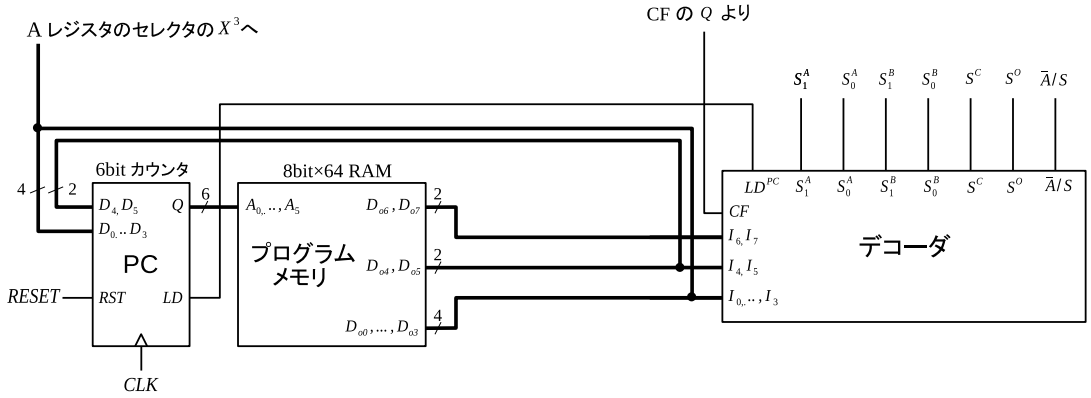
[1] 信号の立ち上がりから次の立ち上がりまでの時間が1周期です。

[2] これにより、この命令 (CPY A, B) の1つ前の命令の実行が完了します。

ただし、1 つの命令に対するこれら 3 段階の処理を完了してから、次の命令のための処理を開始したのでは、全く時間の節約になりませんので、ある命令 STAGE 1 の処理が完了したら、その命令の STAGE 2 の処理を始めるのと同時に、つぎの命令の STAGE 1 の処理を開始します。つまり、実行すべき命令が、たとえば α、β、γ, ... の順に並んでいた場合、次の図のように各命令の各段階を処理します。

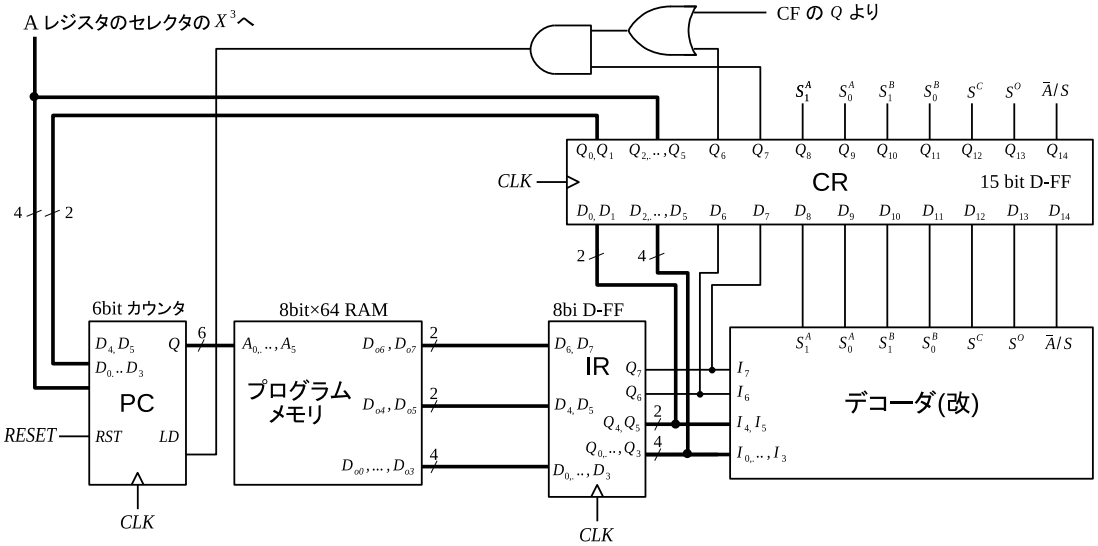


このような命令の実行の仕方を**パイプライン処理**と呼びます。ここで例示したような 3 段階のパイプライン処理は、前回の CPU の回路図の



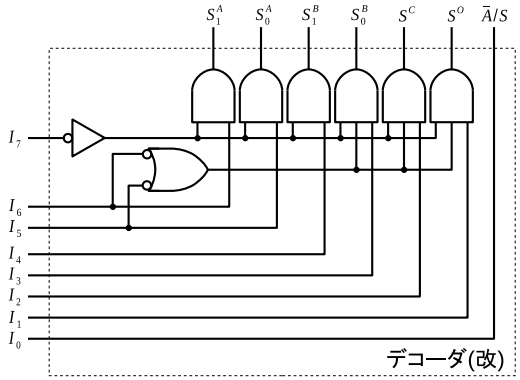
前回の 4 bit CPU の回路図の一部

のようになっていた部分について、次のように、プログラムメモリとデコーダ、デコーダと各セレクタなどの間に (多 bit 長の) D-フリップフロップをそれぞれ挟み込むことによって実現することができます (回路図中の IR と CR)。



(不完全な) パイプライン処理を行う 4 bit CPU の回路図 (一部)

「デコーダ(改)」の部分は、次のような組み合せ回路とします。JPC 命令は最新の CF の値を必要とするため、第 13 回配布資料ではデコーダの中で行われていた LD^{PC} の生成は、CR (15bit D-FF) の出力の後で行われていることに注意してください。

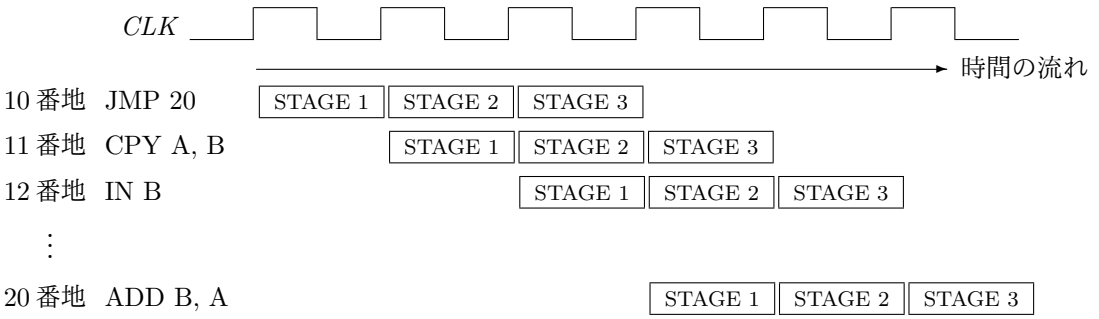


IR や CR の *CLK* にも (他のレジスタと同様に)、この CPU 全体のクロック信号を入力します。各 STAGE の処理をクロック信号の 1 周期の間に完了できればよいので、それだけクロック信号の周期を短く (周波数を高く) することができます。

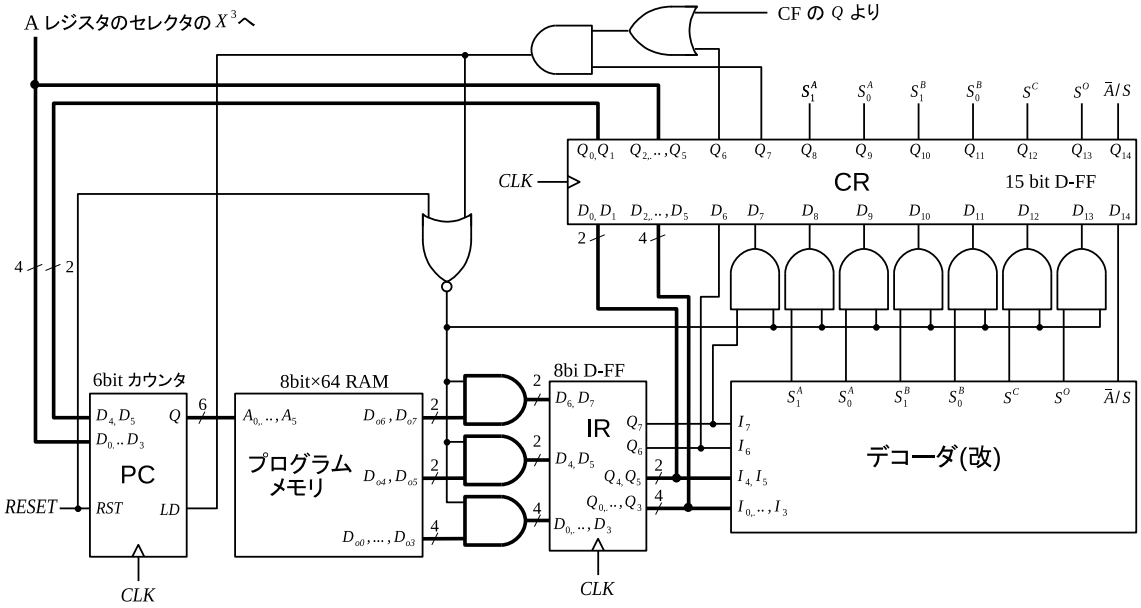
メモ

パイプライン処理における分岐命令の問題　以上のように改良することで、より高速な CPU とすることができそうですが、実は、このままでは、この CPU は正しく動作しません。その原因となるのは分岐命令 (JMP や JPC) です。

分岐命令が実際に効果を持つのは、その分岐命令の STAGE 3 が完了することにより PC (プログラムカウンタ) の値が変更される時です。これにより、変更後の PC の値 (番地) へのジャンプが発生するわけですが、パイプライン処理を行っていると、この分岐命令の直後に置かれている (その分岐が起こらない場合に実行される) 2 つの命令については、それぞれ STAGE 2 と STAGE 1 の処理が完了し、その結果が CR と IR に残こることになります。このため、分岐先の命令が実行される前に、これら 2 つの命令が必ず実行されてしまうことになります。たとえば、10 番地に JMP 20 という命令があり、続いて CPY A, B と IN B という 2 つの命令が、また、20 番地には ADD B, A という命令があった場合の実行の様子は、次の図ようになってしまいます。



この問題を解決するためには、PC の値が変更される際に、それと同時に IR や CR に残る情報を無効化してやります。この無効化は、IR の場合はその値を NOP 命令のビットパターン 00000000 に、CR の場合はその LD^{PC}, S_1^A, S_0^A, S_1^B, S_0^B, S^C, S^O のビットをすべて 0 にしてやる[3]ことで実現できます。次の回路図は、IR や CR の前に AND ゲートを設置し、PC の LD が 1 の場合に、IR や CR の入力を強制的に 0 にすることで、この分岐命令に伴う無効化を実現したものです。また、CPU のリセット ($RESET = 1$) 時も同じ無効化が必要ですので、これも行うようにしています。



(正しく) パイプライン処理を行う 4 bit CPU の回路図 (一部)

計算機システム I ・第 14 回・終わり

[3] これは、NOP 命令をデコードした結果に他なりません。