コンピュータ工学 講義プリント(10 月 9 日)

今回は、ほぼ全てのマイコンに搭載されている周辺機能であるタイマについて説明する。

・タイマの基本構造(教科書 P.33 参照)

タイマは、時間を計るための周辺機能である。タイマを使えば、例えばある時点からの経過時間を計測できたり、一定時間間隔で特定の処理を繰り返したりといった事ができる。最近の高機能マイコンには、複雑な機能を持ったタイマが搭載されているが、教科書で取り扱っている PIC16F84A というマイコンに搭載されているタイマ 0 は、基本的な機能しかなく、理解がしやすいので、このタイマ 0 を題材に、タイマの基本的な機能について説明する事にする。

図 1 が、タイマ 0 のブロック図である。このままでは、まだ理解しにくいので、最も基本的な使い方をする事を想定して、ブロック図を簡略化したのが図 2 である。

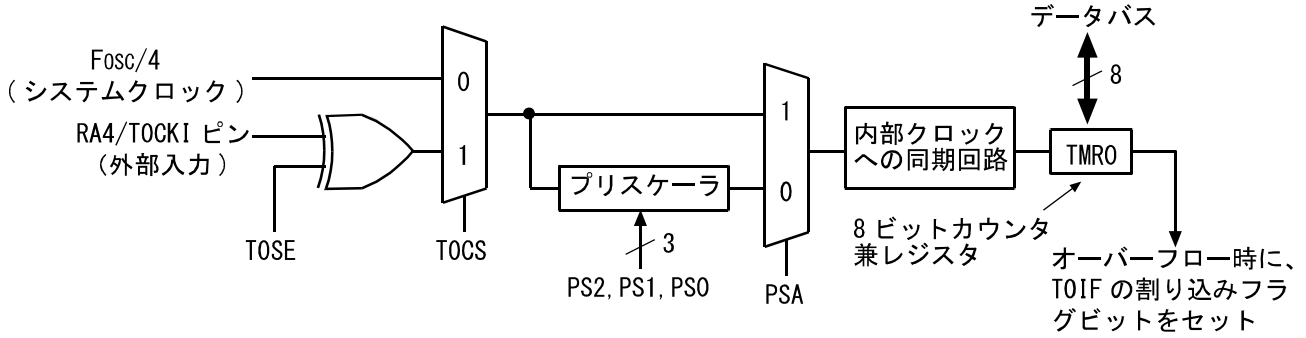

図 1、PIC16F84A のタイマ 0 のブロック図

図 2 を見れば分かるように、タイマの本質はクロック信号のパルスを数えるカウンタ回路である。

図 2 の左側に「Fosc/4(システムクロック)」と書いてあるが、ここにはマイコンを動作させるクロックを入力する。Fosc は、マイコンにつないだクロック発振器の発振周波数である。この Fosc に比例して、マイコンの処理速度が向上する。PIC16F84A の場合は、最大 20MHz のクロック発振器を利用する事ができる。

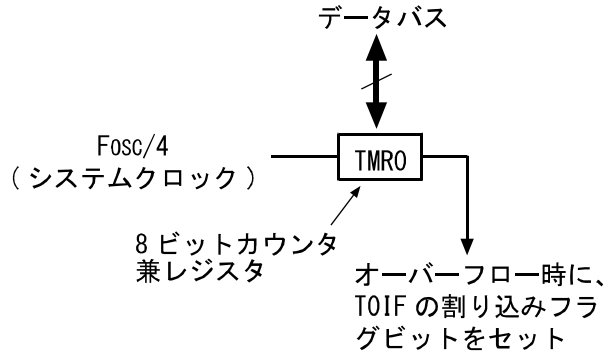

図 2、PIC16F84A のタイマ 0 の簡略化したブロック図 (1)

PIC16 シリーズのマイコンは、クロック発振器の信号を 4 分周してから、システム全体のクロックとして用いる。Fosc/4 というのは 4 分周後のクロック周波数である。

分周というのは、入力されたクロックのパルスを決まった数だけ数えるごとに、出力にパルスを一つ出力する回路を用いる事で、クロックの周波数を整数分の 1 に下げる操作をいう。例えば 4 分周なら、クロックの周波数を 1/4 に下げる。

TMR0 というのは、8 ビットのアップカウンタで、周波数が Fosc/4 のシステムクロックのパルスが一つ入るごとに、数字が一つ増える(これを「カウントアップする」という)。8 ビットのカウンタであるので、0、1、2、…とカウントしてゆき、255 まで数えたら、次にクロックパルスが入ってきたときにはオーバーフローを起こし、0 に戻る。

このTMR0は、カウンタであると同時に、レジスタでもある。データバスにつながっており、CPUがカウンタの値を読み出したり、カウンタの値を書き込んだり(任意の値にセットしたり)できる。

図2の様な構成のタイマで、どのような事ができるか、いくつかの例を見てみよう。

・処理時間の計測

8ビットカウンタTMR0の値がレジスタとしてCPUから読み書きできる性質を利用すれば、例えば特定の処理にかかった時間を計測する事ができる。今、処理時間を計測したい処理を処理Aとよぶ事にすると、処理Aにかかった時間を計測するには、図3のフローチャートに示す様なプログラムを作ればよい。

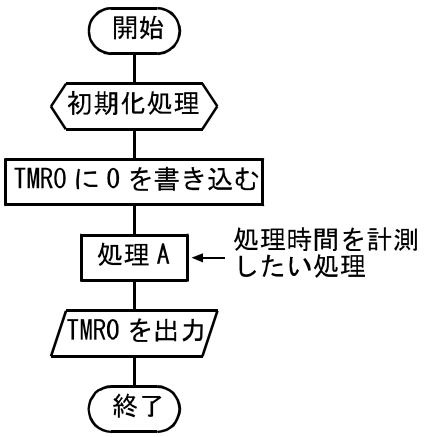

図3、処理Aの処理時間を計測するプログラムのフローチャート

最初に行う初期化処理では、タイマ0の動作モードの設定を含む、各種初期化処理を行う。次にTMR0に0を書き込んで、カウンタをクリアする。次に処理Aを行う。CPUが処理Aを実行している間にも、TMR0は、システムクロックが入ってくるたびにカウントアップしていく。処理Aが終わったら、TMR0の内容を読み出し、適当な表示装置(LEDや液晶など)にその値を出力する。

この様にすれば、処理Aにかかったシステムクロックの数が出力されるはずである。(厳密に言えば、TMR0の読み書きにも時間がかかるので、その補正が必要である)

ただし、TMR0は8ビットカウンタであるので、最大で255クロックまでしか計測できない事に注意が必要である。PIC16F84Aのクロック周波数F_{OSC}を最大の20MHzと仮定して、最大何秒までこの方式で計測できるか計算してみよう。

F_{OSC}が20MHzであるので、それを4分周したシステムクロックの周波数($F_{OSC}/4$)は5MHzである。よって、計測時間の分解能は$1/(5\times10^{6})=0.2\times10^{-6}s=0.2\mu s$となる。さらに、そのクロックを8ビットカウンタで最大255個まで数えられるのであるから、計測できる最大時間は$0.2\times255=51\mu s$(約1万8千分の1秒)である。この様に、かなり短い時間しか測れない事が分かる。これよりも長い時間を計測するには、後述するプリスケーラやタイマ割り込みを併用する必要がある。

・決められた時間の停止

ある処理をしてから、次の処理をするまでに、一定の時間を空けないといけない事は、頻繁にある。

例えば、10文字程度しか表示できない液晶画面に、たくさんのメッセージを表示する場合を考えよう。画面が狭いので、多くのメッセージを一度に表示することはできない。そこで、複数回に分けて順次メッセージを表示することになる。この時に、CPUの処理速度いっぱいでメッセージを次々に表示したら、それを見ている人がメッセージを読めなくなってしまう。よって、あるメッセージを表示したら、次のメッセージの表示までに、CPUを数秒間停止させる必要がある。

また、画面表示の例以外にも、例えば周辺装置に信号を送る際に、信号を送っていい最小の間隔が決まっている場合なども考えられる。この場合も、一度信号を送出してから次の信号を送出するまでに、一定時間CPUを停止させる必要がある。

この様に、決められた時間CPUを停止させる目的にもタイマを使う事ができる。そのためには、図4に示すフローチャートのプログラムを作ればよい。この例では、処理Aと処理Bの間に、約100システムク

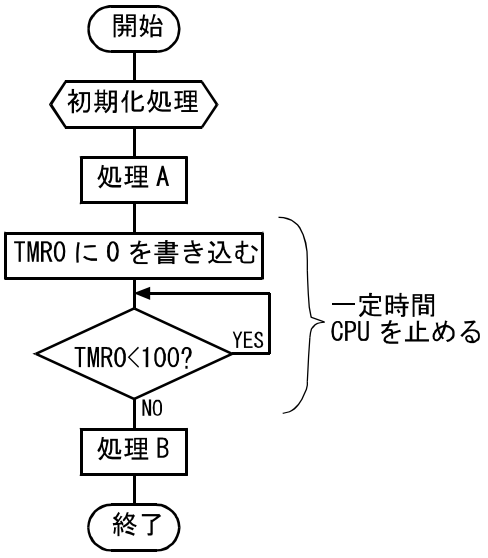

図 4、処理の間に一定時間 CPU を停止する
プログラムのフローチャート (1)

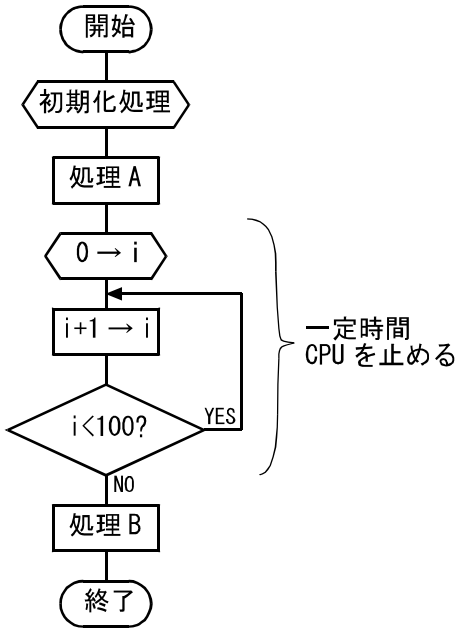

図 5、処理の間に一定時間 CPU を停止する
プログラムのフローチャート (2)

ロック(F_{OSC}=20MHz の場合で 20 μ s)だけ CPU を停止させている。

この例では、CPU を停止させている間、CPU がしている仕事は TMR0 レジスタの内容を確認して、100 システムクロックの時間が経ったかどうかを繰り返し判定する事だけであるので、タイマ機能を使わずに、これを空ループ(ループ中で何もしないループ)に変更しても、同様の事ができる。空ループを使って CPU を停止させる場合のフローチャートを図 5 に示す。

図 5 では、空ループを 100 回回して CPU を止めている。ループを 1 回回すのに数システムクロック必要なので、100 回で数百システムクロック分だけ CPU が停止する事になる。このため図 5 のフローチャートでは、図 4 の場合に比べて、CPU の停止時間が長い。同じ時間だけ CPU を止めるには、ループの回数を減らして調整する必要がある。

図 4 と図 5 に示したように、一定時間 CPU を止める方法には、タイマ機能を使う方法と、空ループを使う方法の 2 通りが考えられる。空ループを使う方が簡便であるようにも思えるが、割り込みを使用するプログラムにおいては、空ループを使う方法では、正確な時間 CPU を止める事ができない欠点がある。なお、割り込みについては、次の講義で説明する予定である。

・インターバルタイマ

一定時間間隔で何らかの処理を繰り返したい事がよくある。例えば、プログラムが動作している事を分かりやすく示すために、LED を一定時間間隔で点滅させる事がよくある。このような場合に、タイマ機能を使わずにプルグラムを書くとすれば、プログラム中の色々なところに分散して LED を点灯または消灯させるコードを埋め込む必要があるし、また、一定時間間隔では点滅できないであろう。この場合はタイマ 0 の割り込みの機能を使えば、問題が解決する。

タイマ 0 の割り込みについて説明する前に、TMR0 の 8 ビットカウンタがオーバーフローした場合に何が起こるかを説明しておく。

TMR0 の 8 ビットカウンタが、255 まで数えてから、さらにクロックパルスが入力されると、オーバーフローが起こり、TMR0 の値が 0 にもどる。もし INTCON レジスタの GIE ビットまたは T01E ビット(表

１参照)の少なくとも一方が 0 ならば、起こるのは、カウンタのオーバーフローだけである。しかし、GIE ビットと T0IE ビットが共に 1 ならば、以下の事が起こる。

カウンタがオーバーフローすると共に、INTCON レジスタの T0IF ビットが 1 になり、タイマ 0 割り込みが発生する。このタイマ 0 割り込みは、一定の時間間隔で発生する。このように、タイマを一定時間経過するごとに合図を発生させる装置として使う場合は、インターバルタイマと呼ぶ。

表 1、タイマ 0 に関係するレジスタの一覧表

アドレス	名称	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
01H	TMR0	Timer 0 Module Register							
0BH	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
81H	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
85H	TRISA	－	－	－	PORTA Data Direction Resister				

注：網掛け部分はタイマ 0 に関係ないビット

割り込みについては、次の講義で説明する予定であるが、タイマ 0 割り込みに限定して簡単に説明しておく。

タイマ 0 割り込みが発生すると、実行中のプログラムが中断し、割り込みサービスルーチン(ISR;Interrupt Service Routine)と呼ばれる、特殊なルーチンが呼ばれる。このルーチンは、RETFIE 命令(ISR からの RETURN 命令、教科書 P.102 参照)を実行すると、元々実行していたプログラムが再開するという意味で、サブルーチンに似ている。RETFIE 命令で、元々のプログラムの処理に戻る前に、ISR 内で T0IF ビットをクリアしておく必要がある。

この ISR の中で、定期的に繰り返したい処理(例えば LED の点灯・消灯)を書いておけば、メインのプログラムの処理にかかわりなく、一定時間間隔で繰り返される処理が記述できる訳である。

タイマ 0 割り込みがどの程度の頻度で起こるかをここで計算しておく。F_{OSC}=20MHz とすると、前述の様に、0.2μs ごとに TMR0 がカウントアップしてゆく。TMR0 が 256 回カウントアップするごとにオーバーフローし、タイマ 0 割り込みが発生するのであるから、タイマ 0 割り込みの発生間隔は 0.2×256=51.2μs となる。また、1 秒あたりの割り込み発生回数は、その逆数の 19531.25 回となる。このように、一秒間に 2 万回近くも割り込みが発生し、メインのプログラムがたびたび中断される事になる。割り込みの頻度を低下させるには、この後で説明するプリスケーラを使えばよい。

・プリスケーラ

OPTION_REG レジスタの PSA ビットを 0 にすれば、タイマ 0 は図 2 に示したブロック図になるが、PSA ビットを 1 にすれば、図 6 に示す様なブロック図となり、8 ビットレジスタの前にプリスケーラが設置された状態になる。

プリスケーラとは、カウンタのオーバーフローまでの時間を延ばす目的で、カウンタの前に設置されるクロック分周器のことである。タイマ 0 のプリスケーラは、OPTION_REG レジスタの下位 3 ビット(PS2、PS1、PS0)で分周比を変えられるようになっている。表 2 に、PS0～PS2 の設定値と分周比の関係を示す。

表 2、PS0～PS2 の設定値とタイマ 0 のプリスケーラの分周比

PS2	PS1	PS0	分周比	PS2	PS1	PS0	分周比
0	0	0	1:2	1	0	0	1:32
0	0	1	1:4	1	0	1	1:64
0	1	0	1:8	1	1	0	1:128
0	1	1	1:16	1	1	1	1:256

このように、カウンタに入るクロックの周波数が、PS0～PS2 の設定により、システムクロックの 1/2～1/256 に低下する。それに伴い、カウンタがオーバーフローするまでの時間が 2～256 倍に延びる。

プリスケーラを使わない場合は Fosc=20MHz の条件でタイマ 0 割り込みの発生間隔が 51.2μs になることは前節で述べたが、プリスケーラを使い、かつ PS0、PS1、PS2 の全てを 1 に設定した場合には、タイマ 0 割り込みの発生間隔が 51.2×256=13107.2μs=13.1072ms にまで延びる。(教科書 P.34 には、「約 26.214ms のタイマを作る事ができます」と書いてあるが、これは Fosc=10MHz の条件で計算しているため、倍に伸びている)

この様に、プリスケーラを使う事で、タイマ 0 の計測時間の上限を伸ばしたり、タイマ 0 割り込みの発生間隔を延ばしたりできるが、時間計測の分解能は、それに伴って低下する事に注意が必要である。したがって、プリスケーラの分周比は、必要な計測時間を確保できる範囲で、なるべく 1:1 に近い分周比を選ぶ必要がある。

・プリスケーラを使っても必要なタイマ時間が得られない場合の対処

インターバルタイマの説明の際に、LED の点滅にタイマ 0 の割り込みが利用できる事を述べたが、最大で 13.1072ms(約 76.3 分の 1 秒)の周期(割り込み発生間隔)しか得られないなら、LED を点滅させても、残像により、人の目には点滅していると感じられないだろう。その様な場合は、ISR 内で点滅周期を伸ばす工夫をすればよい。

図 7 は、おおよそ周期 1 秒で LED を点滅させるための ISR のフローチャートである。

このフローチャートには、cnt と flag の 2 つの変数が出てくるが、これらは共に、8 ビット符号なし整数の変数で、メインルーチンで 0 に初期化されていると仮定している。

「cnt≧38?」の条件判定は ISR が 38 回実行されるたびに 1 度だけ YES になるので、次の「flag=0?」の条件判定が実行されるのは 13.1072×38=498.0736ms(約 0.5 秒)に一度である。また、「flag=0?」の条件判定は、2 度に 1 度 YES となり、2 度に 1 度 NO となる。「flag=0?」が YES なら LED を消灯し、NO な

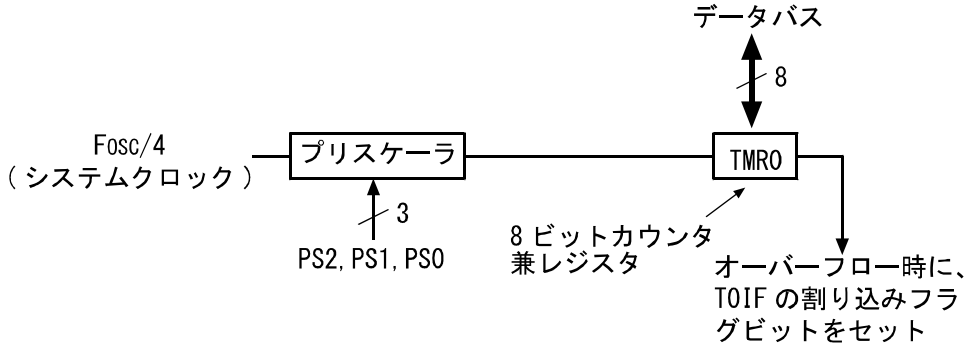

図 6、PIC16F84A のタイマ 0 の簡略化したブロック図 (2)

らLEDを点灯するため、結局498.0736×2=996.1472ms(約1秒)の周期で点滅する。

なお、図7のISRのフローチャートは、簡略化して書いてある。本当は、ISRの開始直後に、Wレジスタと STATUSレジスタの退避をする必要があり、また ISR の終了直前には STATUS レジスタと W レジスタの復帰を行う必要がある。さもなければ、ISR終了後にメインルーチンが暴走する可能性が高い。しかし、これらの退避・復帰処理は、まだ割り込みの説明をしていない時点で説明すると、混乱の元になるので、図7から割愛してある。

ISR内での W レジスタと STATUS レジスタの退避・復帰処理については、次回の講義で説明する予定である。

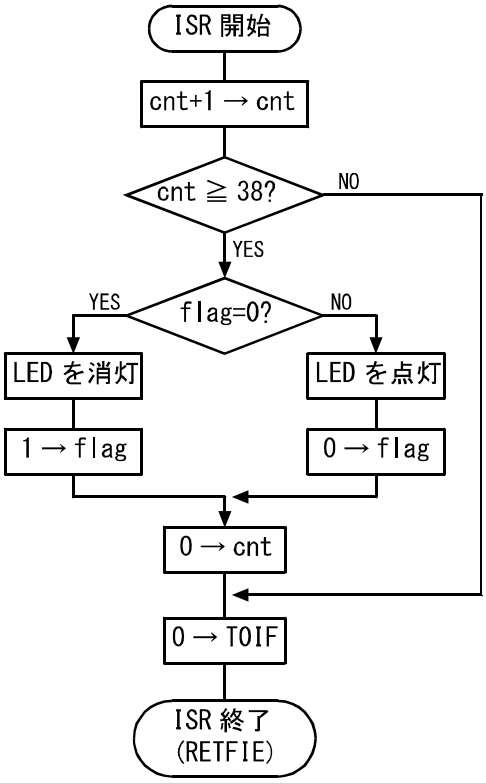

図7、LEDを点滅させるISRのフローチャート

・おわりに

今回は PIC16F84A のタイマ 0 を例に挙げ、タイマの働きや使い方を説明した。PIC16F84A のタイマ 0 には、今回説明した動作モード以外にも、外部クロックでタイマを動作させるモードがある。またタイマ0以外に、CPU が暴走していないかを判定し、暴走時にリセットをかけるためのウオッチドッグタイマも搭載されている。しかし、初学者には複雑すぎる議論になり、また、1回の講義の時間ではこれらについて説明するには短すぎる事から、これらの説明は割愛する。必要に応じて、各自、自習して欲しい。