Ａ君とＣ君は何がなんだか、さっぱりわからなかった。
Ｄ君：「まあ、そのうち、いろいろ勉強していけば、少しづつわかると思うよ。」
なんて言った。

Ｄ君の説明にもう少し付け加えよう。Ｄ君が言ったように、作成する実行可能ファイルはすべてＤＯＳシステムの管理のもとで動作するようにしなければならない。Ｃ言語プログラムはＤＯＳシステムで動作する実行可能ファイルをコンパイラとリンカで作成される。ディスクに保存されている実行可能ファイルは機械語のデータ（機械語の命令データ）だけでなく、ＤＯＳシステム上で動作するための種々の情報も含んでいる。これからみんなが作成するアセンブラプログラムは「Ｃプログラム」の「サブルーチン」として作成するので、マクロアセンブラプログラムは「Ｃ言語」プログラムをコンパイルした場合に得られるアセンブラレベルのプログラムに対応させて作成しなければならない。そうしないと、リンカーによって結合することができない。そのために、マクロアセンブラプログラムにはある決まった形式の構文があるが、これについては必要な箇所のみを次の所で少し説明する。今のところ、アドレスはあまり考えなくてもよいが、先々必ず必要になる。なお、先の「ディスプレイ表示プログラム」はテキストＶＲＡＭのメモリ構成が理解できてないと、わからない。

１９．「基本的なプログラム構成を理解しよう」

ここでは、「Ｃ」と「アセンブラ」のプログラムの基本的な構成と形式を説明する。プログラムはキーボードから入力した２つの数を加算してディスプレイに表示させるものである。ただし、加算演算はアセンブラによって行う。前と同様に「MIFES」で次のようなプログラムを作成する。後の説明の都合上、プログラムに行番号を付けているが、実際のソースプログラムには入れないこと。
「Ｃ」ソースプログラム（ファイル名は SUM.Cとする）

```
1  extern int a,b,c;
2  main()
3  {
4  printf("A="); scanf("%d",&a);
5  printf("B="); scanf("%d",&b);
6  summ();
7  printf("C=%d¥n",c);
8  }
```

「マクロアセンブラ」ソースプログラム（ファイル名は SUMA.ASMとする）

```
1  DGROUP  group   mdata
2  mdata   segment word public 'DATA'
3          assume  ds:DGROUP
4          public  _a,_b,_c
5  _a      DW ?
6  _b      DW ?
7  _c      DW ?
8  mdata   ends
9
```

```
 10  _TEXT    segment byte public 'CODE'
 11           assume  cs:_TEXT,ds:DGROUP
 12           public  _summ
 13  _summ    proc near
 14           MOV  AX,_a
 15           MOV  BX,_b
 16           ADD  AX,BX
 17           MOV  _c,AX
 18           RET
 19  _summ    endp
 20  _TEXT    ENDS
 21           END
```

「Ｃ言語」のプログラミングについては３年の「電子計算機」の講義でいくらか習っていると思うが、プログラム中で使用する「変数」は全て「定義」をしなければならない（ＭＡＳＭ（マクロアセンブラ）プログラミングも同様）。Ｃプログラムで使用する変数は「ａ、ｂ、ｃ」であり、これはＭＡＳＭプログラム中でも使用する。従って、どちらかのプログラム中でその変数の定義をしなければならない。通常、このような変数は「引き数」として受渡しを行うが、ここでは、サブルーチンのプログラムでも同じ変数名で使えるように「グローバル変数」とし、ＭＡＳＭプログラム中で定義する。そのために、Ｃプログラムでは、ｍａｉｎ関数の外で、「ｅｘｔｅｒｎａｌ」宣言する。ここでは、各変数を１６ビット正負表現とするので、「ｉｎｔ」定義を行っている。「ｅｘｔｅｒｎａｌ」宣言とは、「別のソースプログラムで変数が定義されている」ということを示す。

ＭＡＳＭソースプログラムでは、Ｃプログラムの外部変数定義に対して、データセグメント領域でこれらの変数定義を行う（１行～８行）。ただし、これらの変数をＭＡＳＭで定義する時には「＿（アンダーバー）」を付けることになっている。Ｃでは１６ビットで定義しているので、ＭＡＳＭでも対応させて「ＤＷ（Define Word）」で定義する。また、「？」は変数の初期値を定義しない（この場合は「０」で定義ということになるが）ということを表す。プログラム領域とデータ領域は異なるセグメントに定義される。これらのセグメント「ＣＳ」と「ＤＳ」の値はユーザーはわからないが、Ｃ言語では「ＣＳ」は「＿ＴＥＸＴ」、「ＤＳ」は「ＤＧＲＯＵＰ」で仮定されている。そのために「ａｓｓｕｍｅ」宣言でこれらを定義する。さらに、ＭＡＳＭで定義した各変数はＣソースプログラムでも使用されるので、「ｐｕｂｌｉｃ」宣言で「別のソースプログラムでも使用される」という許可を行う。データ領域は「ＤＧＲＯＵＰ」というセグメント、プログラム領域は「＿ＴＥＸＴ」のセグメントである。「ｇｒｏｕｐ」は「ＤＧＲＯＵＰ」セグメントのグループ名を定義している。ここでは「ｍｄａｔａ」としているが、この名称はユーザーが決めてよい。２行目は「ＤＧＲＯＵＰ」の「ｍｄａｔａ」というセグメントの始まりを定義している。このセグメントの終わりを８行目の「ｅｎｄｓ」で定義する。

プログラム領域は「＿ＴＥＸＴ」のセグメントとなるので、１０行目でその始まりの宣言を行う。プログラムはＣからのサブルーチンとしているので、その名称（ここでは「ｓｕｍｍ」としている）の外部参照許可宣言「ｐｕｂｌｉｃ」を１２行目で行う。「ｓｕｍｍ」のサブルーチンの開始を１３行目の「ｐｒｏｃ　ｎｅａｒ」で定義する。「ｐｒｏｃ」は「ｐｒｏｃｅｄｕｒｅ（手続）」の意味であり、「ｎｅａｒ」は「セグメント内のサブルーチン」を表す。１９行目の「ｅｎｄｐ」でプロシジャーの終わりを定義する。なお、ソースプログラムはサブルーチンとしているので、終わりは１８行目の「ＲＥＴ（ＲＥＴＵＲＮ）」である。セグメントの終わりを２０行目の「ＥＮＤＳ」で定義し、全ての終了を「ＥＮＤ」で定義する。

以上が、「Ｃ」と「ＭＡＳＭ」によるソースプログラムの基本的構成である。これから作成するプログラムは、ＭＡＳＭの１４行から１８行目までをプログラミングすることになる。

実際にこれらのプログラムを作成して、コンパイル、リンクを行い実行させてみよう。値を入力して結果を色々調べてみよう。そして、Ａ＝３２７６７とＢ＝１の場合を計算し、その結果が正しいかどうか調べてみよう。それがわかれば、かなり理解していることになるんだけど。

２０．「サブルーチンと割り込みルーチン」

「ルーチン（Routine ）」とはもともと「決まりきった仕事、型にはまったやり方など」の意味である。このサブルーチンに対して、「ＣＡＬＬ（呼び出し）」を行うプログラムをメインルーチンという。先にも述べたように「ＡＳＭプログラム」は「Ｃプログラム」の「ｍａｉｎ関数」から呼ばれる「サブルーチン」であるが、実際は、この「ＡＳＭプログラム」が中心であるので「ＡＳＭプログラム」を「メインルーチン」と考えてもよい。

「サブルーチン」と「割り込みルーチン」の違いを一般的な話で説明してみよう。例えば、自動車で高速道路を走る時、１時間おきにサービスエリアで休憩するような場合、走っている時は速度メータを見たりラジオを聞いたりしている（この状態がメインルーチンに対応）。サービスエリアが見えたら減速する、ウインカーを出す、ハンドルを切る、エリアに入る、休憩する、出発する、という一連の操作を行い、また、通常どおり高速道を走るという繰り返しになる。この時、サービスエリアがある１時間おきにおなじ事を繰り返すことになる。このサービスエリアで休憩することが「サブルーチン」に対応する。これに対し、例えば、一般道路で走行している（メインルーチンに対応）時、急に、交差点で車あるいは人が飛び出して来たときには、とっさにブレーキを踏む、相手の動作に対応して何等かの操作をしないといけない。この場合、いつ、このような操作をするのかはわかってない。このような場合が「割り込みルーチン」に対応する。

実際のプログラムで説明すると「サブルーチン」では次のようである。

MAIN ROUTINE

```
        .
        .
        .
(MAIN PROGRAM)
        .
        .
 CALL SUB1
        .
        .
 CALL SUB1
        .
        .
 CALL SUB1
        .
        .
```

SUB ROUTINE

```
SUB1:   .
        .
        .
     (SUB ROUTINE
      PROGRAM)
        .
        .
        .
        .
      RET
```

「サブルーチン」は「メインルーチン」から「ＣＡＬＬ命令」によって呼び出される。実際は、ＣＰＵの実行アドレスが「ＳＵＢ１」の物理アドレスへ移行する。このよう「サブルーチン」は「メインルーチン」の何処で呼び出されるのかは、わかっており、しかも、その飛び先アドレスは「メインプログラム」の中に書いてあるので、ＣＰＵはそのアドレスに移行する。例えば、サブルーチンがセグメント内の時（ＳＵＢ１はｎｅａｒである）で、そのオフセットが「２０００Ｈ」の時は、メインプログラムの「ＣＡＬＬ」命令は「ＣＡＬＬ　２０００Ｈ」と書いていることと同じである。もう少し詳しく説明すると、命令が「ＣＡＬＬ　ＳＵＢ１」になった時、ＣＰＵはまず、「ＩＰ」に「ＳＵＢ１」のオフセットアドレスを設定する。その時、「サブルーチン」が終了して、再びメインルーチンに戻るためのアドレス（次の命令のアドレス）を「スタック」という所のアドレスに「退避（ＰＵＳＨ）」する。そして、サブプログラムを実行し、「サブルーチン」の終了である「ＲＥＴ」の命令になると、ＣＰＵは先ほど「退避」した「メインルーチン」の開始アドレスを「復帰（ＰＯＰ）」して「ＩＰ」へ取り込み、実行を続けるということになる。「サブルーチン」では通常、「メインルーチン」の内容に関連した処理を行うので、必要のない限り、各レジスタの退避、復帰（割り込みルーチンの所で詳しく説明する）は行わない。実際の制御プログラムでは、値の表示やレベル表示に「サブルーチン」を利用している。ほんとうは、「ＡＳＭプログラム」は「Ｃプログラム」のサブルーチンであるので、サブルーチンからこのような表示用サブルーチンを呼び出していることになる。サブルーチンの基本的な利点は、同じ様な処理を行うプログラムをメインプログラム中に何度も書かずに済むということである。

次に「割り込み」について説明しよう。「割り込み」と言ってもそれには種類が色々ある。「ハードウエア割り込み」、「ソフトウエア割り込み」、「内部割り込み」、「外部割り込み」などであるが、基本的にこれらの割り込みに対するＣＰＵの動作は同じであると考えてよい（「ソフトウエア割り込み」はプログラム構成が異なる）。「メインルーチン」と「割り込みルーチン」の構成は次のようである。

MAIN ROUTINE

```
        .
        .
        .
(MAIN PROGRAM)
        .
        .
        .
        .
        .
```

INTERRUPT ROUTINE

```
IROU:  CLI
       (各レジスタの退避)
          .
          .
          .
          .
       (各レジスタの復帰)
        STI
        IRET
```

卒論で作成するプログラムでは、「ハード外部、あるいはハード内部割り込み」を使用している。「ハード割り込み」とは、ハードウエアによって作成された割り込み信号（負理論）をＣＰＵへ直接（実際は８２５９という割り込みコントローラを通して）送り、ＣＰＵはこの信号を受けたら割り込み処理を行うというものである「割り込みルーチン」も「サブルーチン」と同様に、「メインプログラム」とは別の「割り込みプログラム」がある。先ほどの「サブルーチン」の場合と見比べてみて、不思議な点に気が付けばいいんですが。ここの所は大事だから、Ｄ君がＡ君達３人に説明した話を聞いてみよう。

Ｄ君はいきなり「割り込みといってもそげん難しくないよ」なんて言ったけど、３

人は「そう？」って言って少し真剣になった。（やる気がみられた）
Ｄ君：「まず、ＣＰＵは割り込み信号が入っても、自分自信で割り込みを受け入れられる状態にしていないと、割り込みルーチンに行かないんだ。それを行う命令が「STI(Set Interrupt-enable Flag)」命令たい。だから、メインルーチンではこの命令を実行していないといけない。割り込みルーチンでは、それ以上割り込みを行わないように、最初に「CLI(Clear Interrrupt-enable Flag)」命令を実行し、終わりで、また割り込み許可を行うんだ。割り込みの最後は「ＩＲＥＴ」命令でメインルーチンに戻る」
Ｂ君：「割り込みルーチンの退避とか、復帰とかあるのはなんや？」
Ｄ君：「割り込みというのは、いつ起こるかわからんやろ。普通は、メインルーチンを走っているけど、この時はＡＸとかＢＸなんかのレジスタを使って色々計算なんかをしよるやろ。その時に割り込みルーチンに飛んだときに、割り込みルーチンの中でそれらのレジスタを使うわけたい。そうすると、メインルーチンで計算していた時のレジスタの内容が変わってしまうけん、割り込みルーチンが終わってメインルーチンに戻った時に、値がむちゃくちゃになるやん。だけん、割り込みルーチンで使うレジスタは全部、スタックという所に一時保管するわけ。それを行うことを「退避」といって　「ＰＵＳＨ」命令たい。また、メインルーチンに戻るまえ、要するに「ＩＲＥＴ」を行う前には、それぞれのレジスタを元の値に戻しとかんといかんやろ。それをするのを「復帰」といって「ＰＯＰ」命令じゃ。」
Ｂ君：「ほほー、なるほどね」
Ｄ君：「でも、ここで注意しとかんといかんのは、「ＰＵＳＨ」命令と「ＰＯＰ」命令の実行回数は同じじゃないといかんったい。そうしないと、「ＳＰ」の値がオーバフローを起こして暴走するけんね」
Ａ君：「「ＳＰ」ってなんやったかね？」
Ｃ君：「そんなもん知っとろうも。スタックポインタたい。僕は名前だけしか知らんけど」
Ｄ君：「「ＳＰ」はＰＵＳＨ命令とかＰＯＰ命令のデータ、それとかサブルーチンからの戻り番地なんかを貯めておく時のオフセットアドレスを示すレジスタたい」
Ｂ君：「さっき、不思議なことに気が付けばいいんですが、なんて書いてあったけど、気が付いたばい。あんね、メインプログラム中に割り込みプログラムの飛び先番地が何も書いてないったい。これやろ？」
Ｄ君：「そうたい、いいことに気が付くやんけ」
Ａ君：「すげー。俺、全然わからんやった」
Ｃ君：「Ｂ、おまえ、なんかこっそり勉強しよらんや？」
Ｂ君：「いいや、別に」（でも、本当は勉強していた）
Ｄ君：「割り込みというのは、いつその処理を行うかは決ってないから、プログラム中に書いておくなんてことはできないのさ」
Ａ君：「Ｄ、おまえ、いつから東京弁になったとや」
Ｄ君：（無視して）「だから、あらかじめ割り込みルーチンの開始アドレスを何処かに記憶させておく必要があるのさ。それが、「割り込みベクタ」っていうのさ」
Ａ君：「だけん、その東京弁はやめろって。なんか鳥肌がたってくるけん」
Ｄ君は笑っていた。
Ｄ君：「割り込みっていうのは、実はたくさんあるわけたい。そして、それぞれについてこの「割り込みベクタ」という「アドレス記憶メモリ」があるったい。この「ベクタ」領域は物理アドレス「００００００Ｈ」から割り当てられちょったい。詳しいアドレスは他のマニュアルを見たらわかるけど、僕達が使う

割り込みは、「内部タイマー」というものと「外部割り込みのＩＮＴ５」たい。これらの割り込みに対するベクタ物理アドレスは

内部タイマー：　オフセット　０００２０－０００２１Ｈ
　　セグメント　０００２２－０００２３Ｈ
ＩＮＴ５：　オフセット　０００５０－０００５１Ｈ
　　セグメント　０００５２－０００５３Ｈ

というふうに決まっちょう。だから、これらのアドレスにあらかじめ、これらに対応した割り込みルーチンの飛び先番地をストアしとかんといかんということたい」

Ｃ君：「ふーん、なるほどね」

Ｄ君：「でも、割り込みの場合にはこれだけではすまんで、もっと、色々なことをやらんといかんけど、それはもう少しあとで教えるね」

３人とも幾らかわかったけど「ちょっと、気合いを入れて勉強しないといけないなんて思っていた。

それでは、このもっと色々な事をしないといけないという話をする前に、ハードウエアについての知識がもう少し必要なので、それについて勉強しましょう。

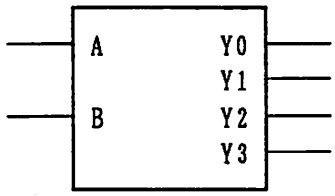
２１．「Ｉ／Ｏアドレスって何なんだ」

ＣＰＵのアドレスバスとデータバスにはプログラムの命令やデータ等を保存するためのＲＡＭあるいはシステム用のＲＯＭが接続されていることは前に説明した通りであるが、さらにこれらのバスにはコンピュータ外部とデータのやり取りを行う入出力装置（Ｉ／Ｏ(Input/Output)）が接続されている。さらに、様々な事を行うためのＬＳＩも接続されている。例えば、入出力装置としては、キーボードやディスプレイあるいは実験で使用するＡ／Ｄ、Ｄ／Ａコンバータなどである。しかし、これらの入出力装置の制御も基本的には、その中に組み込んであるＬＳＩとのデータのやり取りであり、そのＬＳＩにどうやってデータを送ったり、データを取り込んだりするかが問題である。

前述したように、これらの制御ＬＳＩもまた、ＲＡＭと並列にバスで接続されている。制御ＬＳＩにはそれぞれＩ／Ｏアドレスが設定されており、それぞれ異なった番地が割り当てられている。実際は、ＴＴＬデコーダによってＩ／Ｏアドレスを決める。コンピュータ内部のＬＳＩは８ビット（上位８ビットは００Ｈ）アドレスで、拡張スロットなどに組み込むＩ／Ｏ（ＬＳＩ）は１６ビットアドレスとなっている。しかし、ＣＰＵからデータを送る場合に、それがメモリに対するものなのか、ＬＳＩに対するものなのかはバスの信号だけではわからない。そのためにＣＰＵにはメモリとのデータのやり取りなのか、ＬＳＩとのデータのやり取りなのかを知らせるための信号が用意されており、これらの信号をメモリとＬＳＩは調べている。ＣＰＵの命令ではメモリへの読み書きは「ＭＯＶ」命令で行われるが、ＬＳＩに対しては、「ＩＮ」および「ＯＵＴ」命令で行う。

それでは、各ＬＳＩのＩ／Ｏアドレスはどうやって決まるのか。ＬＳＩには通常「ＣＳ(Chip Select)」入力端子があり、この信号が「０」の時に、そのＬＳＩとデータのやり取りが可能となる。「ＣＳ」が「１」の時にはデータバスの信号は何等そのＬＳＩには関係ないということである。この「ＣＳ」信号をアドレスバスのデータに応じて、データがある条件を満たした時にだけ「０」にするような回路が「デコーダ」と呼ばれるものである。簡単な例を示そう。

次の図は２ビットデコーダ（あるいはDemultiplexer:デマルチプレクサ）と呼ばれるものである（例えばTTL:74LS139 ）。



SELECT B	SELECT A	OUTPUTS Y0	Y1	Y2	Y3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

２ビットの入力（アドレスに対応）に対して、その値にによって４つの出力が「０」となる条件が異なる。従って、このＴＴＬでは４つの異なったアドレスが設定できるわけである。実際は８ビットあるいは１６ビットデコードであるので、これに対応させれば、入力が８本あるいは１６本となる。しかし、これでは、例えば入力８本の場合には出力が２５６本となるので、ＴＴＬのピンが多くなり話にならない。通常はアドレスの値がある条件を満たした時だけ「０」とするような論理回路を基本ＴＴＬで組んでデコーダを構成している。

このようなデコーダにより各制御ＬＳＩは異なったＩ／Ｏアドレスをもっている。実際にプログラムを作る時にはこのＩ／Ｏアドレスがわかっていないと何もできないということになる。

２２．「割り込みコントローラ（８２５９）について」

Ｄ君：「さっき話したけど、ハードウエアによる割り込みの場合にはＣＰＵに割り込み信号を送るやろ。その信号っちいうのはこの８２５９という番号の付いている「割り込みコントローラ」を通してくるったい。だけんこの８２５９の働きがわかってないと、割り込みのためのプログラムが組めんとたい」

Ｃ君：「難しいね」

Ｄ君：「図のように８２５９には８個の割り込み信号があり、優先順位があるんたい。そしてそれが２つある。８２５９スレーブの信号はマスタにつなっがている」

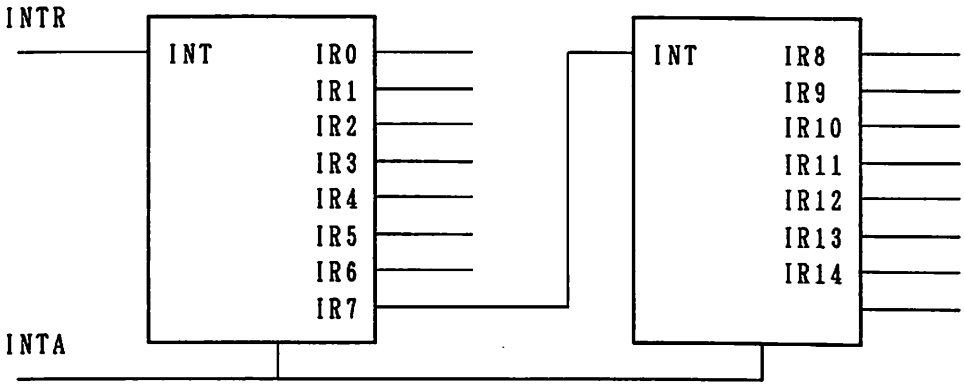
Ａ君：「マスタとかスレーブとか、なんね？」

Ｃ君：「マスタ(master)は主人、スレーブ(slave) は奴隷とかいういみたい。図を見たらだいたいわかるやんか。おまえ英語何点やった？」

Ｄ君：「それぞれの８２５９には各信号を許可、不許可するために８ビットレジスタがあって、それぞれのビットが各信号に対応しちょる。図を書くと

7	6	5	4	3	2	1	0	ビット
IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0	マスタ
	IR14	IR13	IR12	IR11	IR10	IR9	IR8	スレーブ

ビットの値が「０」の時にそれに対応した割り込みだけが許可されるというわけたい。



	マスタ	スレーブ
ＥＯＦアドレス	００Ｈ	０８Ｈ
ＩＭアドレス	０２Ｈ	０ＡＨ

(EOF:End Of Interrupt, IM:Interrupt Mask)

割り込みレベル	デバイス	割り込み要求番号	割り込み名
0	８２５９	IR0	内部タイマー（８２５３）
1	マスタ	IR1	キーボード
2		IR2	ＣＲＴＣ（Ｖ）ディスプレイ
3		IR3	ＩＮＴ０
4		IR4	ＲＳ－２３２Ｃ
5		IR5	ＩＮＴ１
6		IR6	ＩＮＴ２
7		IR7	スレーブ８２５９
8	８２５９	IR8	セントロ・プリンタ
9	スレーブ	IR9	ＩＮＴ３
１０		IR10	ＩＮＴ４
１１		IR11	８”ＦＤ
１２		IR12	ＩＮＴ５
１３		IR13	ＩＮＴ６
１４		IR14	８０８７

実際に使用する割り込みは、内部タイマーだから、これだけを許可するのに、「ＦＥＨ」のデータをマスタの「ＩＭアドレス」に送るんで

ＯＵＴ　０２Ｈ，ＡＬ　　（ＡＬ＝ＦＥＨ）

となるったい。そうすると、タイマー割り込みが入った時にはそれに対応して割り込み信号「ＩＮＴＲ」がＣＰＵに送られる。ＣＰＵがこの信号を受けたら、現在の命令を終了して割り込み応答信号「ＩＮＴＡ（この信号は８２８８を通して送られる）」を８２５９に送るわけたい。そうすると８２５９は、この割り込みに対するベクタアドレスをＣＰＵに出力して、ＣＰＵは割り込みルーチンの開始アドレスを取り込み、割り込み処理を開始するということだ。ＣＰＵは割り込みを終了するときには、再び内部タイマーの割り込みが８２５９マスタによって、受け入れられるように、割り込みが終了したという合図を８２５９マスタに送らないといけないので、つぎのように「２０Ｈ」というデータを８２５９マスタに送っとかないといけない。そうしな

いと、次の割り込みがかからなくなる。

```
OUT  00H, AL    (AL＝20H)
```

これでＣＰＵは通常のメインルーチンを実行して、次の割り込み信号がくるのを待つだけとなるのだ。

Ｂ君：「なるほど。これは難しいね。でも、これがわかってないとプログラムが組めないということやろ」

Ｃ君：「そうたい」

Ｄ君：「もう１つ僕たちが使う割り込み、ＩＮＴ５の場合には、スレーブとマスタを通して割り込み信号が送られてくるので、両方の８２５９に対して同じ命令を実行しないといかんっちゃ。マスタとスレーブはそれぞれアドレスが違うから注意せんといかんね」

Ｃ君：「２つの場合でも、考え方は同じっちゅうことやろ」

Ｄ君：「そう。これだけ理解したら、まあ、卒論でやるプログラムの割り込みについてはだいじょうぶやね。でも、８２５３の割り込み許可のレジスタ内容を書き変えた時には、システムが設定している値を変えるので、予めシステム設定値を保存しておき、終了するときに元に戻すようにしとかないと、暴走の心配があるから気を付けよう」

２３．「割り込み開始アドレスはわかるの？」

だいぶ前の所で、「マクロアセンブラ」ではアドレスを考えなくてもよいということを言ったが、割り込みを使用する場合には、予め割り込みルーチンの開始アドレスを割り込みベクタに書き込んでおく必要がある。従って、割り込み開始の物理アドレス、すなわちセグメントとオフセットを決めなければならない。通常作成する割り込みルーチンは「ＣＳ」内にプログラムするので、そのセグメントは「ＣＳ」と同じである。よって、割り込みルーチンのオフセットアドレスを求めないといけない。ＭＡＳＭではそのために「ＯＦＦＳＥＴ」演算子が用意されている。この演算子によりセグメント内にある変数あるいはラベルのオフセット値を求めることができる。プログラムでは次のように書く。例えば、割り込みルーチンの開始ラベルを「ＩＲＯＵ：」とすると

```
MOV  AX,CS              :CODE SEGMENT
MOV  AX,OFFSET IROU     :OFFSET
```

となる。上の命令ではＡＸにコードセグメントの値が得られ、下の命令では「ＩＲＯＵ：」のラベルがある所のオフセットの値が得られる。これにより割り込みルーチン開始物理アドレスがわかるので、この値を対応した割り込みベクタの所に書き込んでおけばよいということである。ただし、これらの値が最終的に決まるのは、プログラムを実行する時であり、ＤＯＳが管理して行う。

一方、データエリア（「ＤＳ」領域）にある変数のアドレス（オフセット値）を求める場合には、ＤＧＲＯＵＰ内にあるので、次のように書く

```
MOV  AX,OFFSET DGROUP:****     ****は変数名
```

上の形式と

```
MOV  AX,****
```

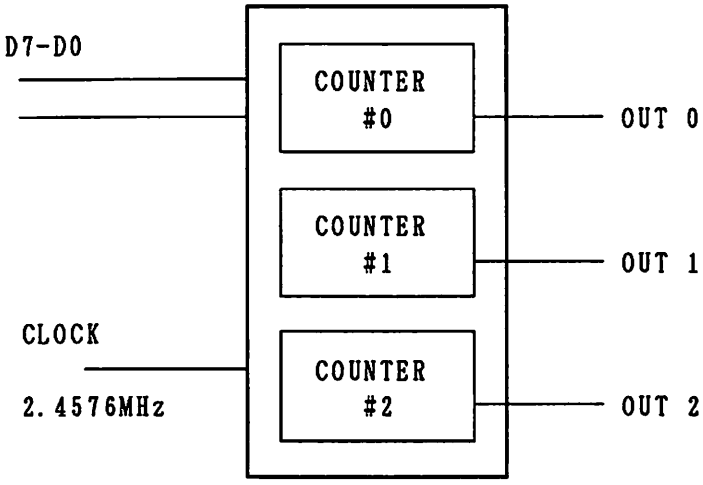
の違いは理解しておかないといけないね。

２４．「内部割り込み用タイマー（８２５３）」

割り込みルーチンを使用する場合のＭＡＳＭプログラム構成はわかったが、実際

に割り込みをかけるにはどうしたらよいか。ここでは、その方法とそれに対するプログラミングを「内部割り込み」について説明しよう。

使用する割り込みは「ハードウエア内部割り込み」である。従って、ＣＰＵ（実際は割り込みコントローラ８２５９）に割り込み信号を送るための何等かの回路（ＬＳＩ）が必要である。そのために、ＰＣ９８０１には内部タイマーがある。これは、８２５３(Programable Interval Timer:PIT)と呼ばれるＬＳＩである。このＬＳＩはカウンタ／タイマーとして動作し、最大４ＭＨｚのクロックで動作する１６ビットカウンタであり、１個のＬＳＩに３個にカウンタ（０、１、２）が内蔵されており、６つの動作モードをもっている。もう少しわかりやすく言えば、プログラムによって与えられたデータの数だけクロックを数えて、そのデータ数になった時に出力を変化するということである。次の図は８２５３の概略図である。



Ｉ／Ｏアドレス
（８ビットデコード）
BASE ADDRESS = 71H

71H	#0 DATA R/W
71H+2	#1 DATA R/W
71H+4	#2 DATA R/W
71H+6	CONTROL DATA W

(R:Read, W:Write)

CONTROL DATA FORMAT

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

COUNTER SELECT

SC1	SC0	
0	0	COUNTER #0
0	1	COUNTER #1
1	0	COUNTER #2
1	1	NOT USED

OPERATIONAL MODES

M2	M1	M0	
0	0	0	MODE 0
0	0	1	MODE 1
X	1	0	MODE 2
X	1	1	MODE 3
1	0	0	MODE 4
1	0	1	MODE 5

READ/WRITE MODES

RL1	RL0	
0	0	COUNT･LATCH OPERATION
0	1	LSB READ/WRITE
1	0	MSB READ/WRITE
1	1	LSB,MSB SEQUENCIAL R/W

COUNT MODES

BCD	
0	BINARY
1	BCD