質問がわからないのか、みんなだまりこけていたが、いきなり

A君：「あのね、まず、最初の「EQU 」っていうのはなんかいな」

C君：「そんなもん簡単やんか、イクオールたい。ようするに、その名前に値を代入するっちいうことたい」

D君：「これは「ＥＱＵディレクティブ」といって、プログラム中で使用する数値をその名前で使用するということを定義しとるったい。「ディレクティブ(Directive) 」ちゃ「指示とか指令」という意味たい。こんなふうに定義しとったら、後で変更する時にそこだけを変えればいいから簡単やんか」

A君：「ほほー、そうやね」

B君：「えっと、なんか、プログラムの最初に「MACRO 」とかいうのがあるんやけど、なんやこれ」

D君：「これは「マクロ」というもんたい。はやい話がプログラム上はサブルーチンみたいなもんやけど、サブルーチンと違うのは、最終的にこのマクロのプログラムはそのままメインプログラムに書いていることと同じになるっちいうことだ。マクロの定義は

```
NAME  MACRO  引き数
        .
        .
      ENDM
```

となる。NAMEは僕たちが勝手に付けれる名前で、引き数が使える。引き数の名前は、マクロ呼出の時のものと同じにする必要はない。普通、引き数は数値だけど、変数名を渡すときには、マクロプログラムのなかでその名前に「&」を付ければいいんだよ。このマクロを呼び出すには、メインプログラムで単に

```
NAME  引き数
```

とすればいいったい。引き数は無くてもいいよ。あとは、プログラムを見ればだいたいわかると思うけど」

C君：「よう、わかった、よっしゃ」（なにか、気合いがはいった）

B君：「だいたい、わかった」

A君：「先に進む前に、なんか「STRUC 」っちいうのがあるけど、これはなに？」

D君：「これは「STRUC ディレクティブ」といって、Ｃプログラムでの構造体に対応する変数の定義たい。まあ、普通の「ＤＷ」とか「ＤＢ」での変数定義と同じように考えとったらよかたい。でも、これで定義した変数の書き方は「名前１．名前２」というふうになるよ」

A君：「なんか、ようわからん」（３人とも首を傾げた）

B君：「マクロとメインプログラムの中に所々、ＢＡＳＩＣみたいな「IF」文があるけど、これはどんな働きがあるっちゃろか」

D君：「この「IF」文はコンパイルの時に機能するったい。要するに、IFの後に書いた変数が０以外の時だけ、「IF」から「ENDIF 」までのプログラムがアセンブルされるわけ。だから、０の時にはそこの所は何も書いてないことと同じたい。何で、こんなもんつけてるかっちいうと、ときどきここだけ実行させたくないとか、一時出力をやめときたいとかいう時に便利やろ。ＭＡＳＭでは「；」でコメントになるけど、いちいちこれを付けたり消したりするのは面倒やんか。だけんたい」

A君：「データ領域に「ＣＲＴデータ」っていうのがあるけど、ここの意味がようわからん」

D君：「これはＤＯＳの内部割り込みっていうのを使って、ディスプレイに色々な

文字をカラーで書かせるためのデータなんだ。データ内容の説明は後にするけど、まず、データの先頭にラベル（ここでは多分 TDATAとなっている）を付けるやろ、そして、メインルーチンで

```
MOV  AH,1
MOV  CL,10H
MOV  DX,OFFSET DGROUP:TDATA
INT  0DCH
```

とすれば、データに書いた文字をディスプレイに表示してくれるわけ。データは「エスケープシーケンス」といって、ＤＯＳシステムが使用している命令なんだ。データはすべて「ＤＢ」で定義し、文字はそのまま「”」で囲めばよい。ここでね、「１ＢＨ」というのが「エスケープキー」の「キャラクタコード」たい。例えば、カーソルをＸ列Ｙ行に動かすにはデータを

```
1BH,"[Y;XH"     (X,Yは数値、ＣＲＴは左上隅が X=1,Y=1
                                  右下隅が X=80,Y=25)
```

とすればよい。このデータの後に文字データを書けばそこからディスプレイに表示する。また、カラーは

```
1BH,"[Cm"
```

とすれば、変わる。このＣの値は

30:黒　31:赤　32:緑　33:黄　34:青　35:紫　36:水色　37:白

で、省略時は白となる。また、４０台はリバースになる。
データの最後は必ず「＄」で終わらせるという決まりがあるので注意しないといけない。まあ、色々書かせてみればおもしろいよ」

Ａ君：「ようわかった。おもしろそうやね」
Ｂ君：「「ＭＯＶ」命令のところでときどき「ＥＳ：」というのが付いているところがあるけど、これはなんやろか？」
Ｄ君：「これは「セグメントオーバーライド演算子」というものたい。通常「ＭＯＶ」命令はメモリ参照の場合には、そのメモリの物理アドレスは「ＤＳ」を使って求めるわけたい」
Ａ君：「ちょっとまって。「ＤＳ」ってなんやったかいな」
Ｃ君：「データセグメントのことたい」
Ｄ君：「この演算子を付けると、その時の「ＭＯＶ」に対してだけ、物理アドレスを作るのに、そのセグメントを使用するということを命令するわけたい。データ領域に定義した変数のセグメントは「ＤＳ」だから問題はないんだけど、例えば、割り込みベクタに書き込む場合なんかは、セグメントを「００００Ｈ」として「ＭＯＶ」命令を行わないといけないし、これから勉強するＤＳＰへデータを送る場合にもセグメントを変えないといけないので、これは覚えとかないといかんよ」
Ｂ君：「そうか、わかった。でもＤ君すごいね。いつ、そんなん勉強したとや」
Ｄ君：「そうないよ。でも、おもしろいやん。自分の作った通りに動くなんて」
Ａ君：「さて、キー入力の所はどうなってるのかなあ」
Ｄ君：「通常、みんなのプログラムはＭＡＳＭサブルーチン（Ｃプログラムから見ると）となっていて、無限ループで実行されているんたい。だから、このループから抜け出すためには、キーボードからの入力等を調べて、ある値が入力されたら抜け出すということを行っているんだ。キーボードの入力はいつもキーボードインターフェイス（８２５１）が調べているから、そのデータを見れば何が押されたかわかるというこったい。このインターフェイスのＩ／Ｏアドレスは「４１Ｈ」なのだ。だから、

```
IN   AL,41H
```

を実行すれば、ＡＬレジスタにキーのデータが得られるので、それを判別す

ればどのキーが押されたかがわかるというしくみだ。キーのデータはキーを押した時（MAKEという）と離された時（BREAK という）でデータが違っている。僕たちは普通、離された時のデータで判別している。通常使うキーの BREAKデータはこのようになっているんだ。

STOP	ESC	TAB	F･1	F･2	F･3	F･4	F･5	F･6	F･7	F･8	F･9	F･10
E0	80	8F	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB (H)

キー入力は簡単なんだよ」

B君：「乗算と除算をちょっと説明してよ」

D君：「８０８６には乗算と除算命令が用意されているんだ。しかも、８ビットでも１６ビットでもいいんだ。じゃあ、まず、乗算をやってみようね。乗算命令はすべてＡＸ（８ビットではＡＬ）を対象に処理される。命令は

MUL reg/mem　（正負表現では　IMUL reg/mem）

である。「IMUL」はInteger Multiplyだよ。８ビット演算なのか１６ビット演算なのかは、reg/mem によって決まるんだ。演算結果は

８ビット演算では、ＡＸ

１６ビット演算では、ＤＸを上位、ＡＸを下位とした３２ビット

となる。乗算はこれだけだ。つぎに、除算では

DIV reg/mem　（正負表現では　IDIV reg/mem）

「IDIV」はInteger Divideという命令なんだ。８ビット除算では

ＡＸ÷(reg/mem)＝ＡＬ、余りはＡＨ

１６ビット除算では

ＤＸ・ＡＸ(32ビット)÷(reg/mem)＝ＡＸ、余りはＤＸ

となる。簡単だね。でも、除算の場合には商が必ず１６ビット（８ビット除算では８ビット）に納まらないといけない。納まらない場合には「INT 0 」の割り込みが発生して暴走してしまうので、注意しないといけないよ」

B君：「以外と簡単やね」

A君：「割り込みプログラムの中で「MOVSW 」という命令があるけど、これは前に出てきた「MOV 」命令と違うんかいな」

D君：「基本的には「MOV 」命令と同じなんだけど、この命令はメモリとメモリ間のデータ転送やブロック転送に使うんだ。「MOVSW (Move String Word)」命令は

［ＤＳ：ＳＩ］→［ＥＳ：ＤＩ］

というふうに１ワードデータを転送する。データを転送した後は、ＳＩとＤＩの値は自動的に２だけ増加されるったい。でも、この値を自動的に増加させるためには、予め「ＣＬＤ(Clear Direction Flag)」命令を実行しておいてディレクションフラグを０にリセットしておかないといけないよ。自動的に減少する場合は「ＳＤＴ(Set Direction Flag)」命令を予め実行しておけばよい。また、バイト単位で転送する場合は「MOVSB 」命令を使う。この命令の時にはＳＩとＤＩの値は自動的に１だけ増加あるいは減少するんだ。さらに、ブロック転送を行う場合には次のようにプログラムすればよかたい。例えば、１０ワードのデータを転送する時は

```
MOV  CX,10
REP  MOVSW
```

たい。すなわち、ＣＸレジスタにデータの数を入れておいて、「MOVSW 」命令に「ＲＥＰ」(Repeat)という「プリフィックス」を付ければいいんだ。バイト単位のブロック転送でも同じだよ。でも、この「ＲＥＰ」を使うより、「MOVSW 」を１０個並べた方が実行時間が速いけどね。まあ、こんなとこかな。この命令はＤＰＳとデータをやり取りする時に使用してるんだ」

C君：「僕はだいたいわかるんだけど、１つだけようわからんところがある。それ

は数値を表示する「DISPA 」とレベルメータみたいな表示を行う「LEVEL 」のサブルーチンなんだ。これはどうなっているのかなあ」

D君：「僕も詳しいことはあまりわからないけど、これらはテキストＶＲＡＭとグラフィックＶＲＡＭというのを使って表示させているんだ。これについては先の所で説明があるから、それを見るとして、簡単にこれらのサブルーチンを使うための方法を説明するね。まず、「DISPA 」からだ。これは、演算した値をそのままディスプレイに表示するものなんだ。しかも、小数点の位置を設定できるから便利だね。ただし、データは正の数として表示するよ。このサブルーチンを呼ぶ前に次のパラメータを設定しておかないといけないよ

```
MOV  AX,(表示データ)
MOV  BL,X            : X座標（左隅が０）
MOV  BH,Y            : Y座標（上隅が０）
MOV  CX,P            : 小数桁数
CALL DISPA
```

ＸとＹはＢＡＳＩＣの「LOCATE X,Y」に対応している。扱っているデータは整数だから小数桁数というのは、ただ単に小数点をどこに入れるかを指定するだけなんだ。例えば、Ｐ＝２の場合でＡＸ＝１の時には「0.01」と表示される。だから、ＡＸ＝１００の時に「1.00」となるので、その時にＡＸの値を「１」に対応させるというだけなんだ。このような考え方はＤＳＰで行う固定小数点演算で利用するので大事なんだけど、まあ、ここではわからなくてもいいや。「DISPA 」の中で行っていることは、表示データを１０進数に変換して（最大５桁６５５３５まで）それぞれの桁の値を「キャラクタコード」に変換してテキストＶＲＡＭに書き込んでいるだけなんだ。

じゃあ、次に「LEVEL 」なんだけど、これはかなり難しいので、どんなことを行うのかだけ話して、あとは今まで使っている「マクロ」を利用することにしよう。「LEVEL 」サブルーチンを利用するには、２つのパラメータを設定しないといけない。それはＡＸレジスタにどれだけのドット数（ディスプレイは縦が４００ドットである）だけレベルを上げるかという値を入れとく。サブルーチンは最大で１４０Ｈ（３２０）ドット表示するようになっているから、それに対応するように値を決めるんだ。次に、ＤＩレジスタに０レベルのところのドットアドレスを入れるんだけど、これは、グラフィックＶＲＡＭの構成がわかってないと決められないよ。だから、ここでは、そのアドレスを求める計算式を教えるね。ディスプレイの一番上が０ドット目で一番下が３９９ドット目となっている。また、横方向は１カラム単位（一番左８ドットが０、一番右８ドットが７９）の設定となっているので、例えば、左からＸカラム目、上からＹドット目を０レベルにするための、ＤＩに与えるアドレスは

ＤＩ＝（Ｘ－１）＋（Ｙ－１）＊５０Ｈ

となる。ただし、レベルの幅は２カラム（１６ドット）使うので、Ｘ≦７８の範囲、レベルは最大３２０ドットとなるので、Ｙ≧３２０の範囲て設定しないと、おかしな表示になる場合があるよ。だけん、実際のプログラミングは次のようになるよ。

```
MOV  AX,(レベルデータ)
MOV  CS:ALEV,AX
MOV  AX,NAME
MOV  CS:BLEV,AX
MOV  DI,(レベルアドレス)
CALL LEVEL
MOV  AX,CS:BLEV
```

```
MOV  NAME,AX
```

ここでね、「NAME」はデータ領域に定義した１ワードの変数で、たくさんのレベルを表示させる場合にはそれぞれについて違う変数名で定義しておかないといけないんだ。レベルは黄色で表示されるけど、プログラムを解読すれば自分の好きな色に変えることだってできると思うよ。
それから、これらのサブルーチンは実際に作成するプログラムファイルの中に入れて置くんじゃなくて、別のファイルに入れておいて、コンパイルするときにソースファイルに入れるようにするんだ。そのために、「INCLUDE ディレクティブ」を使って、ソースプログラムのセグメント内（通常ソースプログラムの最後）で次のように、その命令を書いておけばいいよ。

DISPAを使用する場合　　INCLUDE B:¥DISPA.INC
LEVELを使用する場合　　INCLUDE B:¥LEVEL.INC

上の形式はそれぞれのサブルーチンソースファイル「DISPA.INC 」と「LEVEL.INC 」がドライブＢの親ディレクトリにある場合だから、子ディレクトリに保存している場合にはディレクトリ名も指定しないといけないからね。普通、これらのインクルードファイルは子ディレクトリ「INCLUDE （含むという意味）」に入れておくとわかりやすいね。
これらのサブルーチンは少し難しいけど、結構役に立つから、ソースプログラムを見て解読するのもいいと思うんだけど」

Ｃ君：「そうだな。やってみよう。プログラムの勉強にもなるし。ＶＲＡＭの構成もわかるからね」
Ａ君：「これでだいたいプログラム全体の命令がわかったような感じだね。あとはそれぞれの命令はマニュアルなんかを見て調べればいいんじゃないの」
Ｂ君：「そうだね。あとはみんなのやる気しだいだよ。じゃあ、これでディスカッションを終わりにしよう。ああ、疲れた」

３２．「ＶＲＡＭっておもしろいよ」

ＶＲＡＭ（ビデオＲＡＭ）とはディスプレイに文字やグラフィック図を書くために用意されているＲＡＭであり、ディスプレイ画面と１対１に対応したメモリである。ＶＲＡＭのデータはＧＤＣ（グラフィック・ディスプレイ・コントローラ）とＣＲＴＣ（ＣＲＴコントローラ）のＬＳＩによって制御され、ＶＲＡＭにデータを書き込むだけで、ディスプレイ表示ができる。Ｎ８８ＢＡＳＩＣに慣れている人は知っていると思うが、ディスプレイ表示には「テキスト画面」と「グラフィック画面」がある。これに対応して、ＶＲＡＭは「テキストＶＲＡＭ」と「グラフィックＶＲＡＭ」があり、それぞれ別のＲＡＭによって構成されている。

それでは、まず、「テキストＶＲＡＭ」を説明しよう。「テキストＶＲＡＭ」すなわち、「ＡＮＫ(Alphabet Numeral Kana) 用ＶＲＡＭ」は画面上の１つのキャラクタ（通常の画面では８０×２５キャラクタ）に１対１に対応し、ＶＲＡＭは表示エリアとアトリビュートエリア（キャラクタの色などの情報を入れる所）で構成され、しかもこれらのメモリを２画面分もっている。通常、使用しているのはその内の表示エリア１画面であり、アトリビュートエリアは操作していない。この表示エリアと画面との対応は次のようになっている。

すなわち、セグメントがＡ０００Ｈで、オフセットが０００００Ｈ－０Ｆ９ＦＨの範囲に割り当てられている。奇数番地は日本語表示の場合に使用する。ＡＮＫ文字だけの場合は図のように、偶数番地のみである。ここに書き込むデータはキャラクタコードである。すなわち、例えば、「１」という数字を表示したい時には、それに対応したキャラクタコード「３１Ｈ」を書き込む必要がある。

行＼桁	0	1	2	3		77	78	79
0	A0000	A0002	A0004	A0006	···············	A009A	A009C	A009E
1	A00A0	A00A2	A00A4	A00A6		A013A	A013C	A013E
2	·							·
·	·							·
行	·							·
·	·							·
23	A0E60	A0E62	A0E64	A0E66		A0EFA	A0EFC	A0EFE
24	A0F00	A0F02	A0F04	A0F06	···············	A0F9A	A0F9C	A0F9E

アトリビュートエリア（A2000-A2F9E）
もう１画面のエリア（表示：A1000-A1FFE
　　　　　　　　　　アトリビュート:A3000-A3FFE）

キャラクタコードは次のようである。

上位４ビット（横）／下位４ビット（縦）

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		DE		0	@	P		p				ー	タ	ミ		×
1	SH	D1	!	1	A	Q	a	q			。	ア	チ	ム		円
2	SX	D2	"	2	B	R	b	r			「	イ	ツ	メ		年
3	EX	D3	#	3	C	S	c	s			」	ウ	テ	モ		月
4	ET	D4	$	4	D	T	d	t			、	エ	ト	ヤ		日
5	EQ	NK	%	5	E	U	e	u			・	オ	ナ	ユ		時
6	AK	SN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		分
7	BL	EB	'	7	G	W	g	w			ァ	キ	ヌ	ラ		秒
8	BS	CN	(	8	H	X	h	x			ィ	ク	ネ	リ		
9	HT	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A	LF	SB	*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	HM	EC	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C	CL	→	,	<	L	¥	l	|			ャ	シ	フ	ワ	●	
D	CR	←	−	=	M	]	m	}			ュ	ス	ヘ	ン	○	
E	SO	↑	.	>	N	^	n	~			ョ	セ	ホ	゛	／	
F	SI	↓	／	?	O	_	o				ッ	ソ	マ	゜	＼	

01H-1FH はコントロールキャラクタ
80H-9FH, E0H-EBH はグラフィックキャラクタ
例えば、Ｎ８８ＢＡＳＩＣで直接このＶＲＡＭにデータを書き込んで画面に表示させるには、次のようにすればよい。（命令の意味はマニュアル参照）

```
DEF SEG=&HA000
POKE &H9E,&H41
```

これを実行すると、画面の右上隅に「Ａ」という文字を表示する。同じことをアセンブラで行うと次のようなプログラム（書き方は色々あるが）となる。

```
MOV   AX,0A000H
MOV   ES,AX
MOV   AX,41H
MOV   BX,9EH
MOV   ES:[BX],AX
```

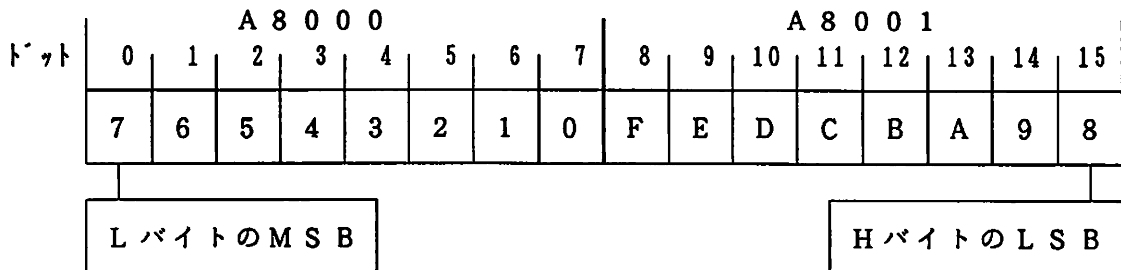
すなわち、書き込む番地（画面の位置）を決めて、そこに表示したい文字のキャラクタコードを書き込むということである。前の「DISPA 」サブルーチンは、このようにして、数字を表示している。

次に、「グラフィックＶＲＡＭ（ＧＶＲＡＭ）」を説明しよう。これも「テキストＶＲＡＭ」と同様にメモリとＣＲＴがそのまま対応しているが、「ＧＶＲＡＭ」ではメモリ内のビット１つ１つがＣＲＴ上の１ドットに対応している。ＧＶＲＡＭには４つのメモリがあり、それぞれGVRAM0(青(Ｂ))、GVRAM1(赤(Ｒ))、GVRAM2(緑(Ｇ))、GVRAM3(インテンシティ)（GVRAM3は通常使用しない）となっている。それぞれにデータを書き込めばそれに対応した色でビットデータが「１」の所に対応したドットが表示されるということである。この３つGVRAM への書き込みを組み合わせれば合計８種類の色で表示できるわけである。各GVRAM のメモリ構成は同じであり、その物理アドレスが異なるだけである。次の図はGVRAM0(青)のメモリ構成とＣＲＴドットの関係である。

0------------------------ ドット ------------------------639

0---ドット--15　16--ドット--32

0	A8000	A8001	A8002	A8003	・・・・・・・	A804E	A804F
1	A8050	A8051	A8052	A8053		A809E	A809F
・ ドット ・							
398	AFC70	AFC71	AFC72	AFC73		AFCAE	AFCAF
399	AFCB0	AFCB1	AFCB2	AFCB3		AFCFE	AFCFF



他のGVRAM も同様な構成であり、開始アドレスは次のようになっている。

GVRAM1(赤): B0000H　　GVRAM2(緑): B8000H

色の組合せは次のようである。

	黒	青	赤	紫	緑	水	黄	白
GVRAM0(B)	0	1	0	1	0	1	0	1
GVRAM1(R)	0	0	1	1	0	0	1	1
GVRAM2(G)	0	0	0	0	1	1	1	1

例えば、Ｎ８８ＢＡＳＩＣで直接このＧＶＲＡＭにデータを書き込んで画面に表示させるには、次のようにすればよい。（命令の意味はマニュアル参照）

```
DEF SEG=&HB800
POKE &H0,&HFF
```

これを実行すると、画面の左上隅に「緑色の８ドットライン」が表示される。これと同じことをアセンブラで行うと次のようなプログラム（書き方は色々あるが）となる。

```
MOV  AX,0B800H
MOV  ES,AX
MOV  AX,00FFH
XOR  BX,BX
MOV  ES:[BX],AX
```

すなわち、書き込む番地（画面のドット位置）を決めて、そこに表示したいデータを書き込むということである。前の「LEVEL 」サブルーチンは、このようにして、グラフィック表示している（黄色の場合にはGVRAM1とGVRAM2のエリアに同じデータを書き込めばよい）。

アセンブラによるこのようなＶＲＡＭを直接アクセスする表示法をマスターすれば、ＢＡＳＩＣやＣによるプログラムよりもはるかに高速に色々な文字や図形を描くことが可能である。

３３．「除算のテクニック（ビット操作）」

まず、レジスタのビット操作について簡単に説明しよう。前に出てきた８２５３のモード設定や先で出てくるＤＳＰとのハンドシェイクなどを行う場合には、レジスタのあるビットだけを「０」にするとか「１」にする（この時他のビットは変えない）ということが必要となる。こういった操作を行うために「論理演算命令」がある。すなわち、ＡＮＤ（論理積）、ＯＲ（論理和）、ＥＸＣＬＵＳＩＶＥ ＯＲ（排他的論理和）、ＮＯＴ（否定）である。これらの論理演算はよく知っていると思うが、これらの操作に対する命令はそれぞれ、「ＡＮＤ」、「ＯＲ」、「ＸＯＲ」、「ＮＯＴ」である。また、２の補数を求める（正負を反転させる）命令として「ＮＥＧ」が用意されている。「ＮＥＧ」命令は「ＮＯＴ」を行って「ＩＮＣ」をすることと同じである。

それでは、この節のタイトルである「除算のテクニック」について述べよう。どうしてタイトルに「乗算」が含まれていないかと言うと、先で出てくるＤＳＰの命令には「除算」命令がないからである。８０８６ＣＰＵでは除算命令「MUL(IMUL)」があるので、この命令を使えばいいんじゃないかと思うかも知れないが、ここで説明するのは、この除算命令を使わずに除算を行う一つの方法である。その前に、２進数についてもう一度考えてみよう。例えば「１０」は２進数で

　　１０Ｄ＝０００１０１０Ｂ

である。また、「２０」は２進数で

　　２０Ｄ＝０００１０１００Ｂ

となる。この２つをよく見ると「２０」の表現は「１０」の２進数表現全体を左に１ビットだけずらしたものとなっている。すなわち、１ビット左にずらせば「２倍」になるわけである。従って、２ビット左にずらせば

　　４０Ｄ＝００１０１０００Ｂ

「４倍」、３ビット左にずらせば「８倍」となる。要するにＮビット左にずらせば「２のＮ乗倍」となる。これは、２進数が２の累乗で表されているので当然である。この時、一番下位のビット（ＬＳＢ）には「０」が入ってくることに注意しておこ

う。１０進数で「１０倍」するには桁を左に１つずらして「１００」とすることと同じである。
このように考えると、逆に右に１ビットずらせば「１／２倍」になるということも理解できるでしょう。すなわち、

５Ｄ＝００００１０１Ｂ

である。さらに、この「５」を右に１ビットずらすと

００００００１０Ｂ＝２Ｄ

となる。この時、「５」の半分は「２．５」であるが、小数部は表せないので切り捨ててしまうから、その誤差に注意が必要である。また、一番上位ビット「ＭＳＢ」には「０」を入れていることに注意しよう。このようにビットをずらす（このことを「シフト」という）ことで「２の累乗の乗除算」はできるということである。
いまは正の数の場合を考えたが、負の数ではどうだろうか。たとえば「－１０」は２進数で

－１０Ｄ＝１１１１０１１０Ｂ

である。これを左に１ビットシフトすると

－２０Ｄ＝１１１０１１００Ｂ

で、２倍となる。「２のＮ乗倍」する時も正の数の場合と同じである。この時もＬＳＢには「０」をつめていることに注意しよう。それでは、「－１０」を１／２倍するために右に１ビットシフトすると

０１１１１０１１Ｂ

となる。正の数と同じようにＭＳＢに「０」を入れるとこれは「＋１２３」となってしまう。「－１０」の１／２は「－５」でありこれは

１１１１１０１１Ｂ

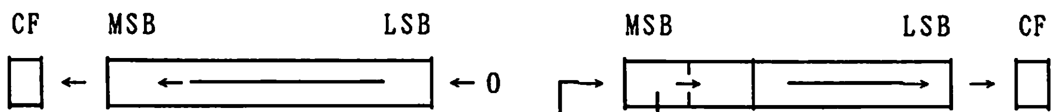
である。すなわち、負の数の「１／（２のＮ乗）」は必ず負の数となるので、ＭＳＢは「１」としないといけない。このことは正の数の場合と違っている。
このようにしてさらに、「－５」を右に１ビットシフトすると

１１１１１１０１Ｂ

となり、これは「－３」である。実際は「－２．５」である。そうすると正の数の時とは少し違うことに気が付くでしょう。すなわち、ビット操作で得られる整数値はＢＡＳＩＣにおける「ＩＮＴ（その値を越えない最大の整数値）」命令を実行していることと同じである。
これらをまとめると

「２のＮ乗」を得るには「Ｎビット左シフトしてＬＳＢに０をつめていく」
「１／（２のＮ乗）」を得るには「Ｎビット右シフトし
　　扱う数値が正の数の時はＭＳＢに０をつめる
　　扱う数値が正負表現の時はＭＳＢの値は変えない」
　　（正の数の時もＭＳＢの値は変えないと考えてよい）

ということになる。この操作を行う命令が「算術シフト命令」と呼ばれるもので、次のようである。



ＳＡＬ命令
(Shift Arithmetic Left)

ＳＡＲ命令
(Shift Arithmetic Right)

この命令の書き方は、

```
SAL  reg/mem,1      SAR  reg/mem,1
SAL  reg/mem,CL     SAR  reg/mem,CL
```

第２オペランドは１ビットシフトの時は「１」、複数ビットシフトの時は「CLレジスタの値」で指定できるが、複数ビットシフトの時は１ビットシフト命令をその数だけ並べた方が処理時間が短縮できる。ただし、プログラムステップ数は増加する。以上は８ビットについて説明したが、１６ビットの場合でも同じである。

ここで、図の中に「ＣＦ」というのがあるが、これは「キャリーフラグ」といって、加算や減算命令などで桁上げや桁下がりが起こった時に「１」にセットされるフラグである。シフト命令でこのＣＦを使っているのは、３２ビットで算術シフトを行うためである。３２ビットの場合はどうすればよいか各自で考えてみよう。

この算術シフト命令に対して、「論理シフト」命令がある。これはただ単にビットをシフトさせる命令であり、次のように処理される。

CF ← [MSB ←―― LSB] ← 0　　　0 → [MSB ――→ LSB] → CF

ＳＨＬ命令(Shift Left)　　　　ＳＨＲ命令(Shift Right)

命令の書き方は算術シフト命令と同じである。「ＳＨＬ」命令は「ＳＡＬ」命令と全く同じ処理を行う。この他に「ローテート」命令があるが、これはあまり使ってないので、説明は省略する。

２の累乗の乗除算はシフト操作で簡単に行うことができ、「乗除算命令」を行うよりはるかに速く処理できるが、左シフト（乗算）の場合には扱っている数値の大きさに注意していないといけない。

２の累乗の計算の場合には前述したような方法で求めることができるが、２の累乗でない乗算を行う場合には「MUL(IMUL)」命令を使用するか、あるいは２の累乗のシフト操作で構成したプログラムを作成（例えば、ＡＸの５倍の値を求める時には「ＡＸ＊４＋ＡＸ」など）しないといけない。ＤＳＰでは「乗算命令」は１マシンサイクル（ＣＰＵが命令を実行するためのクロック周波数）で実行されるので「乗算」に対してはそのまま「乗算命令」を使用しても演算時間の問題はない。しかし、ＤＳＰの命令には「除算」がないため除算用のプログラムを組まないといけない（通常３０ステップ位）。したがって、乗算命令を用いて除算するプログラムテクニックを行い、実行時間の短縮を図る。その一つの方法を説明する。ＤＳＰに対する命令はまだ、勉強していないので、８０８６命令で説明しよう。この方法は卒論の実際の制御プログラムで頻繁に使用しているので、理解すること。

簡単な例として、ある値（８ビットで考える）を「５」で除算する場合を説明する。基本的な考え方は「除算」を「乗算して除算する」ということである。「乗算して除算する」なら初めから「除算」すればいいと考えればそれまでだが、この「乗算して除算する」の「除算する」が重要な所である。すなわち、この「除算するを前に説明したシフト操作（最終的にはシフト操作も何もしないことになる、すなわち除算を乗算だけで行うことになるが）で行うわけである。

「５」で割るということは、「０．２」を掛けるということである。「０．２」は「２を掛けて１０で割る（あるいは４を掛けて２０で割るなどと考えてもよく、いくらでもその値は考えられる）」であるが、これでは乗数も除数も２の累乗でないから、乗除算命令を使わないといけない。これでは意味がない。そこで、除数を２の累乗にして、対応する乗数を求める。それで、いま除数を「６４（２の６乗）とすると、乗数は64/5=12.8 となるので「１３」とする。この値で、乗算を行うとＡＬ＝１００の時、

ＡＸ＝１００＊１３