８．アセンブラ命令

ＣＰＵが解読できるマシン語の１つ１つに対応してプログラマがわかり易いように記述した命令がアセンブラ（Assembler）である。アセンブラ命令は，先にも述べたように

［ニーモニック］　［第１オペランド］，［第２オペランド］

の形式で記述し，「ニーモニック（mnemonic code）」は命令機能を表し，「オペランド（operation code）」は命令に対するデータやメモリを表す。通常，オペランドに対する操作は「第１オペランド」を基準にして行われ，命令によっては「第１オペランド」だけのもの，あるいはオペランドを必要としないものがある。ある処理を行うプログラムは利用するアセンブラ命令によって様々な記述法があり，最も効率的な（プログラムステップ数の少ない方法，処理速度の速い方法）プログラムがよいプログラムであるといえる。

ここでは，現在の制御プログラムで使用している命令について説明するが，その他については付録に示しており，詳細はマニュアルを参照されたい。

［データ転送命令］

レジスタ間，レジスタとメモリでデータ転送（Move）あるいは数値をレジスタまたはメモリに転送する命令である。

```
MOV (reg/mem/sreg),(reg/mem/sreg/imm)
```

「reg」は８あるいは１６ビットレジスタ，「mem」はメモリ（変数），「sreg」はセグメントレジスタ（１６ビット），「imm」は数値（即値）であり，第２オペランドから第１オペランドへデータが渡される。また，２つのオペランドのビットは一致しなければならない。

即値を転送する場合をイミディエットモード（immediate mode）といい，例えば次のように記述する。

```
MOV AX,1234H            または  MOV CL,10H  など
```

レジスタ内容を転送する場合をレジスタアドレッシング（register addressing）（直接モード）といい，例えば次のように記述する。

```
MOV AX,DX               または  MOV BH,AL  など
```

メモリ内容をレジスタに転送する場合をメモリアドレッシング（memory addressing）（間接モード）といい，格納するメモリのアドレスを指定する。アセンブラ記述ではアドレスはほとんどの場合，変数としてデータ領域で定義し，その変数名を記述する。

```
MOV BX,DATAW            または  MOV AL,DATAB  など
```

また，後述するＤＳＰやデータテーブルあるいはＶＲＡＭ等，あらかじめアドレスがわかっているメモリへのアクセスはつぎのように記述する。

```
MOV AX,[BX]             または  MOV DX,[SI]  など
```

なお「[ ]」内のレジスタにはあらかじめ指定するアドレスを格納しておく。このモードでアドレスの指定に使用できるレジスタは「BX,BP,SI,DI」であり，他にも幾つかの記述方法

がある。メモリのデータ転送ではその物理アドレスを作成するために通常「ＤＳ（データセグメント）」が使われる。しかし，ＶＲＡＭ等，そのセグメントで作成できない物理アドレスのメモリ間でデータを転送する場合があり，その時には次のように記述し，その命令に限って指定したセグメントを使用することを指示する。

MOV AX,ES:[BX]　　または　MOV DX,ES:[SI]　など

「ES（エクストラセグメント）」にはあらかじめ所望のセグメントを入れておく。これにより，この「MOV」命令だけメモリの物理アドレスを「ES」で作成してデータを転送する。このような記号を「セグメントオーバーライドプリフィックス（Segment Override Prefix）」といい，「CS:,DS:,ES:,SS:」がある。

また，データテーブルなどを使用する際には，データ領域内のそのデータテーブルの格納アドレス（通常はその先頭番地であり，アセンブラではそこにラベルを付ける）が必要になる。この場合にもセグメントオーバーライドプリフィックスを指定するが，そのアドレスを求めるために「OFFSET ディレクティブ（Directive）」を用い，

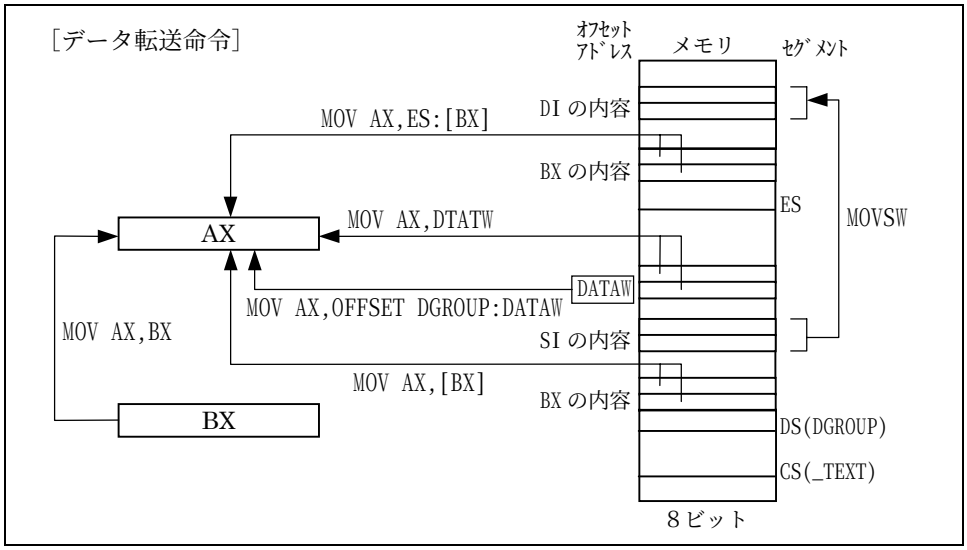
MOV AX,OFFSET DGROUP:ﾗﾍﾞﾙ

のように記述する（Ｃプログラムの DS は DGROUP で仮定されている）。なお，「DGROUP」を省略した場合には「CS」とみなされ，プログラム領域のラベルに対してそのオフセットアドレスを求めてレジスタに格納する。

データをメモリ上のアドレスから別のアドレスへ転送する場合にはストリング命令（String Instruction）による転送命令を用いる。この命令はブロック転送などに利用でき，幾つかの命令がある。実際のプログラムで使用するのは

MOVSW

で，この命令は「DS:SI」で示されるアドレスの１ワードデータを「ES:DI」で示されるアドレスへ転送し，その後，「SI」「DI」レジスタの値をそれぞれ２増加（「CLD」命令によりディレクションフラグが０にクリアされている場合）するものである。



［加減算命令］

レジスタ間，レジスタとメモリおよび即値で加算（Add），減算（Substract）を行う命令である。

(ADD,SUB) (reg/mem),(reg/mem/imm)

２つのオペランドに対して演算を行い，その結果は第１オペランドに格納される。その際，第２オペランドの内容は変化しない。実行時には正値表現か正負表現かを考慮し，オーバーフロー（桁上げや桁下がり）に注意が必要である。なお，これらはＣＰＵ内のフラグレジスタによって表され，オーバーフローを起こした場合には「キャリーフラグ（Carry Flag）」と呼ばれるビットを「１」とするようになっている。これを考慮した加減算命令もある。フラグレジスタの内容はこのような演算命令や比較命令によって変化し，フラグにはこの他にも色々あるが（付録の命令参照），必要に応じて説明する。

加減算命令にはその内容を「１」だけ増加（Increment）あるいは減少（Decrement）するものがあり，次のようなニーモニックである。

(INC,DEC) (reg/mem)

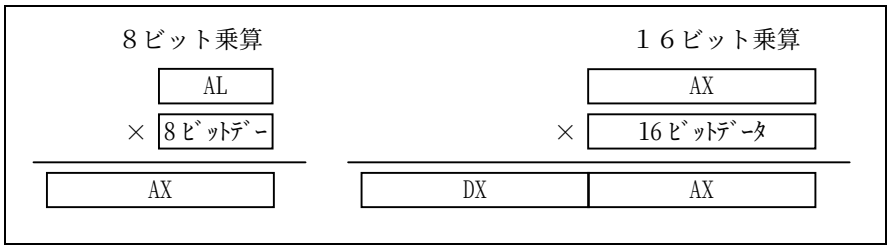
レジスタあるいはメモリ内容をそのまま「１」増加あるいは減少する。

［乗除算命令］

乗算（Multiply）命令は次のように記述する。

(MUL,IMUL) (reg/mem)

「MUL」は符号なし（正値表現），「IMUL」は符号付き（正負表現）乗算（Integer Multiply）である。いずれも演算は８ビット乗算は「AL」レジスタに対して行われ，結果は「AX」レジスタに格納され，また，１６ビット乗算では「AX」レジスタに対して行われ，結果は「DX」（上位１６ビット）「AX」（下位１６ビット）レジスタに格納される。８ビット演算であるか１６ビット演算であるかは，オペランドのタイプで決まる。



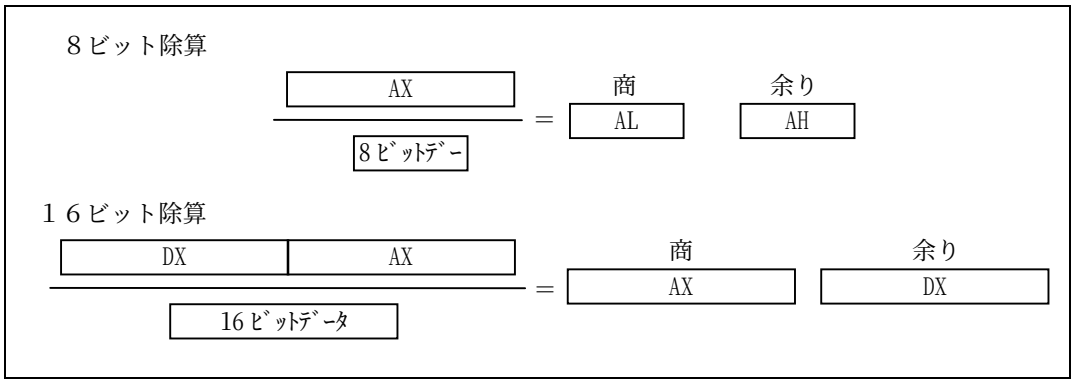
一方，除算（Divide）命令は次のように記述する。

(DIV,IDIV) (reg/mem)

「DIV」は符号なし（正値表現），「IDIV」は符号付き（正負表現）除算である。いずれも除数８ビット除算の時は被除数は「AX」レジスタであり，結果の商は「AL」レジスタ，余りは「AH」に格納され，また，除数１６ビット除算では被除数は上位１６ビットが「DX」

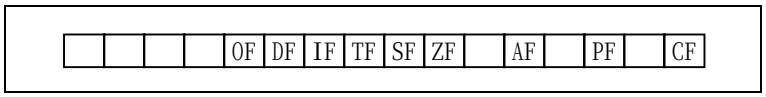
下位１６ビットが「AX」レジスタで表現される３２ビットであり，結果の商は「AX」，余りは「DX」レジスタに格納される。８ビット演算であるか１６ビット演算であるかは，オペランドのタイプで決まる。

なお，除算では商が必ず８ビット（AL）あるいは１６ビット（AX）となるように被除数と除数を与えなければならない。商が指定レジスタで表現できる値の範囲を越える場合には，「INT0」と呼ばれる割り込みが発生し，実行を停止する（プログラムの暴走）するので，注意が必要である。



［フラグと制御命令］

ＣＰＵ内部には演算などよって得られた結果に対して，その状態を示すためにフラグレジスタがあり，レジスタ内部はそれぞれ独立したビットで構成されている。ユーザーは直接これらのフラグを操作することはほとんどないので，ここでは使用しているものだけについて簡単に説明する。



ＣＦ（キャリーフラグ：Carry Flag）
演算の結果，桁上げ（キャリー）または借り（ボロー）が生じた時に「１」がセットされる。
ＺＦ（ゼロフラグ：Zero Flag）
演算の結果，レジスタあるいはメモリ内容がゼロとなった時に「１」となる。
ＳＦ（サインフラグ：Sign Flag）
演算の結果，最上位ビットが１の時「１」，ゼロの時「０」となる。
ＯＦ（オーバーフローフラグ：Overflow Flag）
符号数値とみなして演算を行った結果，オーバーフローあるいはアンダーフローを生じた時に「１」がセットされる。
ＤＦ（ディレクションフラグ：Direction Flag）
ストリング操作を行う際に，アドレスを自動的に増加させる時「０」，減少させる時「１」となる。
ＩＦ（インタラプトイネーブルフラグ：Interrupt Enable Flag）
ＣＰＵに対して外部からの割り込みを許可するか否かを決定する。「１」の時に割り込みを許可する。

ディレクションフラグを「１」にセット（アドレスを自動的に減少）する。

```
STD                                    (Set Direction Flag)
```

ディレクションフラグを「０」にリセット（アドレスを自動的に増加）する。

```
CLD                                    (Clear Direction Flag)
```

インタラプトフラグを「１」にセット（割り込み許可）する。

```
STI                                    (Set Interrupt-Enable Flag)
```

インタラプトフラグを「０」にリセット（割り込み禁止）する。

```
CLI                                    (Clear Interrupt-Enable Flag)
```

［比較命令］

２つのレジスタ，メモリおよび即値に対して，その内容を比較（Compare）する命令である。

```
CMP (reg/mem),(reg/mem/imm)
```

この操作により，フラグレジスタ内容が変化する。この命令は基本的には，前述の「SUB」と同じ機能であるが，第１オペランドの内容は変化しない。

［分岐命令］

プログラムの実行を指定のアドレスへジャンプ（Jump）するものであり，「無条件分岐」と「条件分岐」命令がある。「無条件分岐」は次のようである。

```
JMP (reg/mem/imm)
```

分岐先のアドレス指定方法はいくつかあるが，通常は同一セグメント内の直接指定であり，アセンブラ記述では，ラベルによって例えば次のように記述する。

```
JMP ﾗﾍﾞﾙ
```

「条件分岐」とは，その分岐命令が実行される時のフラグ内容を調べて，その状況に応じて分岐するか否かを行う命令である。なお，分岐しない時は次の命令を実行する。条件分岐命令にはいくつかあり，次のように記述する。

```
ニーモニック ﾗﾍﾞﾙ
```

条件分岐命令では相対アドレス分岐を行うので，その分岐範囲は「-128～127」バイト内に限られる。分岐を表すニーモニックは以下のようである（A,B の大小関係は「CMP A,B」で比較した場合である）。

A,B の大小関係	符号なし数値の場合	符号付き数値の場合
A＞B	ＪＡ　／ＪＮＢＥ	ＪＧ　／ＪＮＬＥ
A≧B	ＪＡＥ／ＪＮＢ	ＪＧＥ／ＪＮＬ
A＝B	ＪＥ　／ＪＺ	ＪＥ　／ＪＺ
A≦B	ＪＢＥ／ＪＮＡ	ＪＬＥ／ＪＮＧ
A＜B	ＪＢ　／ＪＮＡＥ	ＪＬ　／ＪＮＧＥ
A≠B	ＪＮＥ／ＪＮＺ	ＪＮＥ／ＪＮＺ

ニーモニック内の記号の意味は「A:Above」，「B:Below」，「G:Greater」，「L:less」，

「Z:Zero」，「E:Equal」，「N:Not」である。また，「／」で区切られた２つの表現は全く同じ関係を別表現したものであり，アセンブラではどちらを使用してもよい。

［論理演算命令］

論理演算とは「AND」（論理積：AND），「OR」（論理和：OR），「NOT」（否定：NOT）および「EXCLUSIVE OR」（排他的論理和：XOR）をビット単位で行う命令である。

```
(AND,OR,XOR) (reg/mem),(reg/mem/imm)
```

```
NOT (reg/mem)
```

また，フラグ内容だけを変化させる命令として，

```
TEST (reg/mem),(reg/imm)
```

があり，これは基本的に「AND」操作を行うが，レジスタ，メモリ内容は変化しない。さらに，正負表現数値に対してその２の補数（符号反転）を求める次の命令がある。

```
NEG (reg/mem)
```

［シフト・ローテート命令］

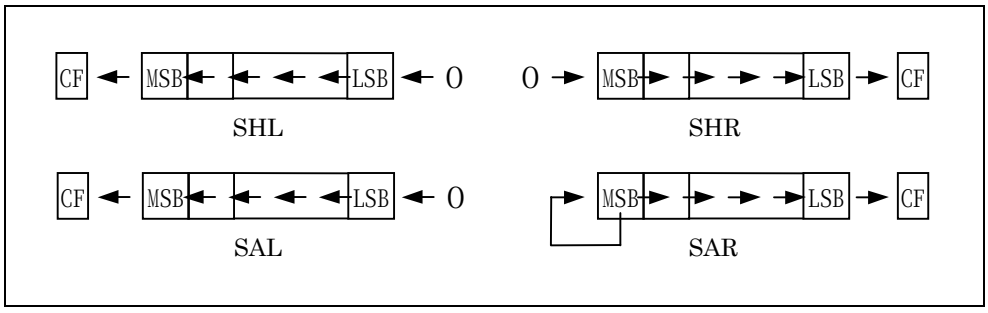
これらの命令は８あるいは１６ビットをまとめて，その内容を移動（Shift）あるいは回転（Rotate）するものである。後者はあまり使用していないので，ここではシフト命令を示す。この命令の記述は次のようである。

```
(SHL,SHR,SAL,SAR) (reg/mem),1
```

あるいは

```
(SHL,SHR,SAL,SAR) (reg/mem),CL
```

前者の命令では１ビットシフトし，後者の命令では「CL」レジスタの示すビット数だけシフト（「CL」レジスタにはあらかじめシフト数を格納しておく）する。記号は「R：Right（右）」，「L：Left（左）」を表す。命令の操作内容は以下のとおりである。



「SHL」と「SAL」は全く同じ操作を行う。「SHL,SHR」を「論理シフト」といい正値表現の場合などに使用し，「SAL,SAR」を「算術シフト」とよび，正負表現の場合に利用する。

［入出力命令］

入出力（Ｉ／Ｏ：Input/Output）命令はキーボードやコンピュータに接続した外部入出力装置とのデータのやり取りを行うものである。外部Ｉ／Ｏにはそれぞれポート番号と呼ばれるＩ／Ｏアドレス（８ビットあるいは１６ビット）が決められており，そのアドレスを指定してデータの入出力を行う。命令の記述は入力命令が

```
IN (AL/AX),ﾎﾟｰﾄ番号
```

あるいは

```
IN (AL/AX),DX
```

出力命令は

```
OUT ﾎﾟｰﾄ番号,(AL/AX)
```

あるいは

```
OUT DX,(AL/AX)
```

のように記述する。ポート番号で直接指定する場合にはＩ／Ｏアドレスは「00H～FFH」（８ビットアドレス），「DX」レジスタでポート番号を指定する場合にはＩ／Ｏアドレスは「0000H～FFFFH」（１６ビットアドレス）である。データが８ビットであるか１６ビットであるかは，オペランドの「AL」あるいは「AX」で決められる。

［ループ命令］

プログラムのある部分を繰り返して実行するために用いられ，単純ループ命令と条件付きループ命令がある。単純ループ命令の表現は次のようである（後者についてはマニュアルを参考にされたい）。

```
LOOP ﾗﾍﾞﾙ
```

「LOOP」命令は，あらかじめ「CX」レジスタに格納された値をディクリメント，すなわち「－１」し，その結果が「０」でなければ「ラベル」へジャンプする操作を行う。実際のプログラムでは時間待ちやディスプレイの表示などで使用している。

［サブルーチン・割り込み関係の命令］

これらについては後で詳しく説明するので，ここでは簡単に命令を説明する。「サブルーチン（Sub Routine）」とは１連の処理を行う別のプログラムのことをいい，メインプログラム（ここではＣで定義したＡＳＭプログラムであるが，厳密にはＡＳＭプログラムもサブルーチンである）から次の命令によって実行される。

```
CALL ｻﾌﾞﾙｰﾁﾝ名
```

サブルーチンには名前を付け，ＡＳＭプログラムでは「proc near」～「endp」のプロシジャーでソースプログラム内に作成する。「CALL」命令によってプログラムの実行はサブルーチンへ移る。サブルーチンからメインプログラムへ戻る時には

```
RET
```

を実行する。「RET」は「Return」である。したがって，サブルーチンの最後の命令は必ず「RET」である。

サブルーチンから別のサブルーチンを呼び出すこともできるが，サブルーチンの呼び出しは通常メインルーチンであり，メインプログラム中に「CALL」が記述される。ところが，メインプログラム実行中に外部から信号が来た時に特別な処理を行うような場合があり，この特別な処理を行うプログラムを「割り込みプログラム（Interrupt Program）」といい，このようなプログラムの実行を「割り込み処理」という。割り込みでは，いつその処理を行うかは決まっていないので，メインプログラムから呼び出すわけにはいかず，何等かの準備（割り込みが受け入れられるような状態）をしなければならない。割り込みプログラムも１種のサブルーチンであり，ＡＳＭソースプログラム中ではサブルーチンと同様に記述する。ただし，割り込みルーチンからメインプログラムへ戻る場合には

```
IRET
```

を実行する。「IRET」は「Interrupt Return」である。したがって，割り込みルーチンの最後の命令は必ず「IRET」である。

サブルーチンでは通常，メインプログラムに関連した処理を行うが，割り込みルーチンではほとんどの場合，メインプログラムとは無関係な処理を行う。したがって，メインルーチンで使用したレジスタは，割り込みルーチンでも使用し，その内容は保存されない。そのため，割り込みルーチンではレジスタ内容を一時，別のメモリ領域（スタック（Stack）と呼ばれる領域）に保存し，メインルーチンへ戻る時にそれらの内容を元に戻す操作を行う。レジスタ内容をスタックへ保存（これを「退避：Push」という）する命令は

```
PUSH (reg16/sreg/mem16)
```

である。また，レジスタ内容をスタックから戻す（これを「復帰：Pop」という）命令は

```
POP (reg16/sreg/mem16)
```

である。

マクロアセンブラでは BASIC と同じように「ラベル」を使用することができ，メモリアドレスをほとんど意識せずにプログラミングができる。オペランドにはラベルをそのまま記述するが，ジャンプ先のラベルには最後に「：（コロン）」を付ける決まりになっており，ラベルは英文字で始まらなければならない。なお，ラベルはデータ領域で定義する変数とは異なるので，注意されたい。