# 第10章 x86系CPUのデータ転送命令



本章から，386以降の32ビットのx86系CPUで使用可能な命令セットである，Windowsのアセンブラ MASMと，Linuxのアセンブラgasでの実際の記述法を示しながら，その動作内容を解説します．

## 10.1 32ビットx86系CPUの命令の種類

32ビットのx86系CPUの機械語命令は大きく，
(1) 汎用命令
(2) FPU命令
(3) SIMD命令
(4) システム命令
に分けられます．

汎用命令とは，OSから一般のアプリケーションまで広く使われる命令で，演算は整数で行います．

FPU命令は，浮動小数点演算に関する命令です．演算は2進浮動小数点で行われ，四則演算や平方根，三角関数，指数関数といった演算が行えます．

SIMDは，複数の整数や2進浮動小数点を同時に演算するための機能で，そのための命令セットがSIMD命令です．SIMD命令は，CPUの発売時期によって使える機能が異なるため，CPUに依存する命令といえます．

システム命令は，その名が示すようにシステム，つまりCPUを制御するための命令で，OS本体やOS周りのプログラム作成に使用します．通常のアプリケーション・プログラムでは，システム命令は使用しません．CPUの保護機構によってシステム命令は，一般アプリケーションからは実行できないようになっています．

### 10.1.1 CPUの新しい命令にアセンブラは対応していないことがある

x86系CPUは，新しいCPUが発売されるたびに新しい命令が追加されてきました．一般ユーザが，この追加された新しい命令を使う場合，アセンブラが新しい命令に対応している必要があります．

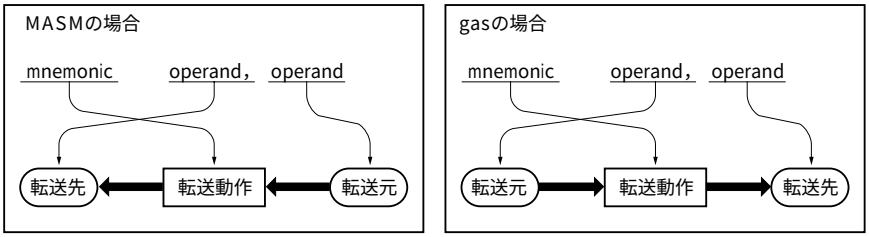
しかし，アセンブラのバージョン・アップはどうしても遅れがちで，新しい命令への対応は，CPU発売後になり，相当の時間を必要とする場合があります．本書で使用しているアセンブラも，386CPUレベルの命令はサポートしているものの，バージョンによりPentiumで追加された命令，なかでもSIMD命令の使用には制約があるようです．

そのため，せっかくCPUに新しい命令があっても，アセンブラの制約からその新しい命令が使えないということもあるので，新しい命令をアセンブラで使う場合にはとくに注意が必要です．

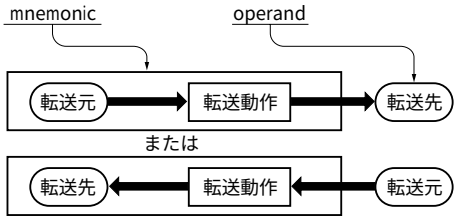
### 10.1.2 汎用命令は11種類

汎用命令は，Intel社のマニュアル(IA-32インテルアーキテクチャ・ソフトウェアデベロッパーズマニュアル)によると，
(1) データ転送命令
(2) 2進算術命令
(3) 10進算術命令
(4) 論理命令
(5) シフト命令とローテート命令
(6) ビット命令とバイト命令
(7) 制御転送命令
(8) ストリング命令
(9) フラグ制御命令
(10) セグメント・レジスタ命令
(11) そのほかの命令
の11種類に分類されています．

そこでまず，整数データの転送命令について説明します．




（**a**）オペランド二つで転送元と転送先を指定する場合




（**b**）ニモニックと一つのオペランドで転送元と転送先を指定する場合

**図1**
**ニモニックによるデータ転送元と転送先の指定**

## 10.2 整数データ転送命令の記述のしかた

整数データの転送では，転送元(source)と転送先(destination)が必ず指定されます．転送命令は，転送元の値をリードし，転送先にその値をライトします．そのとき，転送元の値は変化しません．また，データ転送の命令は一部の命令を除き，フラグへは影響を与えません．

整数データの転送では，転送元としてはイミディエイトやCPUのレジスタ，メモリ，I/Oポート，実効アドレス(オフセット)が指定できます．また，転送先としてCPUのレジスタ，メモリ，I/Oポートが指定できます．

整数データ転送で使用されるサイズは，8ビット長のバイト，16ビット長のワード，32ビット長のダブル・ワードの3種類です．

アセンブラ上の記述では，ニモニックによって転送元と転送先の二つのオペランドを指定するものがあります．また，ニモニック自体が特定の転送元あるいは転送先を決めている場合があり，その場合は転送元あるいは転送先のみをオペランドで指定します(**図1**)．

### 10.2.1 レジスタに対する転送の制限

86系CPU自体は，汎用レジスタやセグメント・レジスタ，フラグ・レジスタ，そしてシステム・レジスタに対する転送命令をもっています(**図2**)．

一般的なアプリケーション・プログラムでは，汎用レジスタやセグメント・レジスタ，フラグ・レジスタの一部が転送の対象となります．しかし，システム・レジスタは，一般的なアプリケーション・プログラムでは使用できません．

システム・レジスタやフラグ・レジスタの一部は，システム命令での転送となり，OSレベルのプログラムで使用され，CPU起動時の初期化や例外処理，OSが管理する仮想記憶の制御などに使用されます．そのため，OS上で実行される一般的なアプリケーション・プログラムがこのシステム・レジスタにアクセスした場合，例外を発生することによってCPUがシステム・レジスタ上の値を保護しています．

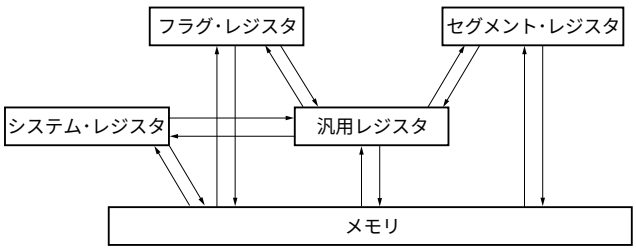



**図2**
**データ転送の対象**

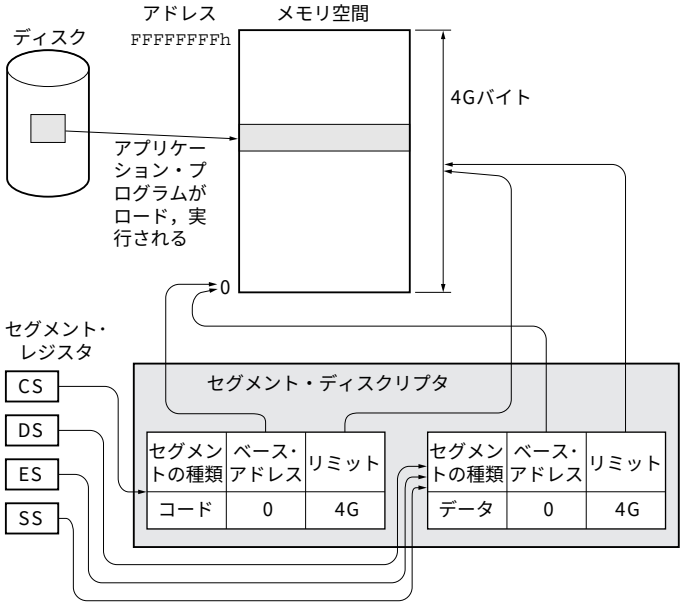



**図3**
**WindowsやLinuxでアプリケーションの実行に使用されるセグメント**

また，x86系CPU上で実行されるWindowsやLinuxのアプリケーションでは，セグメント・レジスタもOSが管理しています．そのため，セグメント・レジスタも一般的なアプリケーションでは転送先に指定することができません．

x86系CPU上で実行されるWindowsやLinuxは，アプリケーション・プログラムに対して，**図3**のような大きなセグメントを一つ割り当てます．この一つの大きなセグメントの領域の一部に，アプリケーション・プログラムのコードとデータを一緒にロードして実行します．

そして，CPU自体は，

　セグメントの先頭アドレス＋オフセット値

でアドレスを求め，メモリをアクセスします．しかし，アプリケーション・プログラムのレベルではセグメントの値は考えず，32ビットのオフセットのみでメモリをアクセスすることになります．

したがって，WindowsやLinuxのアプリケーション・プログラムでは，セグメント・レジスタの内容は変化させないよう，またセグメント・レジスタを転送先に指定しないような注意が必要です．

### 10.2.2　メモリに対する転送の条件

今述べたように，x86系CPU上で実行されるWindowsやLinuxのアプリケーション・プログラムは，オフセットのみを使用してメモリにアクセスします．

オフセットは32ビットあり，バイト単位でアドレスが振られているため，32ビット・オフセットなら4Gバイトのメモリ空間がアクセスできることになります．この4Gバイトのメモリ空間にはアプリケーション・プログラムのほかに，アプリケーション間で共通に使用されるライブラリやOS本体なども配置されています(**図4**)．

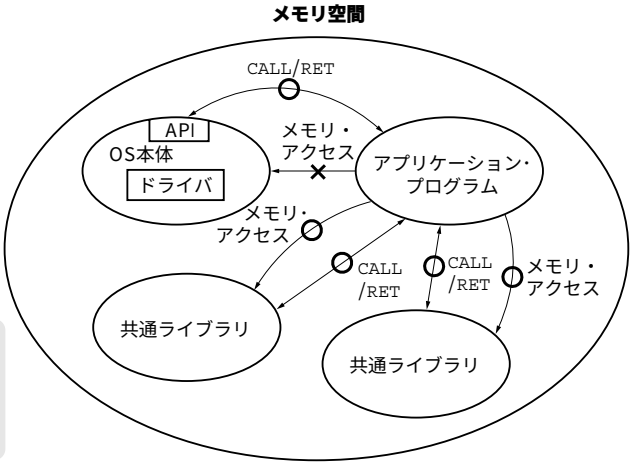
メモリにアクセスする場合，バイトはもちろんワード，ダブル・ワードでも，自由なアドレスに配置できます．たとえば，バイト，ワード，ダブル・ワードを**図5**のように隙間なく連続して配置することができます．

ただし，ワードやダブル・ワードのような複数のバイトで一つの値を表すデータは，配置のしかたによっては物理的なメモリのアクセスを倍に増やしてしまい，実行速度を落とす原因となる場合があります．

CPUに接続されている物理的なメモリは，1回のアクセスで複数バイトのリード/ライトを行うことができます．たとえば，1回の物理的なアクセスで，386なら32ビット(4バイト)，Pentiumは64ビット(8バイト)をアクセスする能力をもっています．

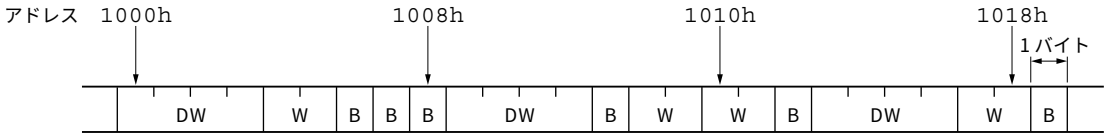
そのため，ワードやダブル・ワードの先頭アドレスによっては，一つのワードやダブル・ワードのアクセスが，2回の物理的なアクセスとなる場合があります(**図6**)．

これを防ぐためには，ワードやダブル・ワードの



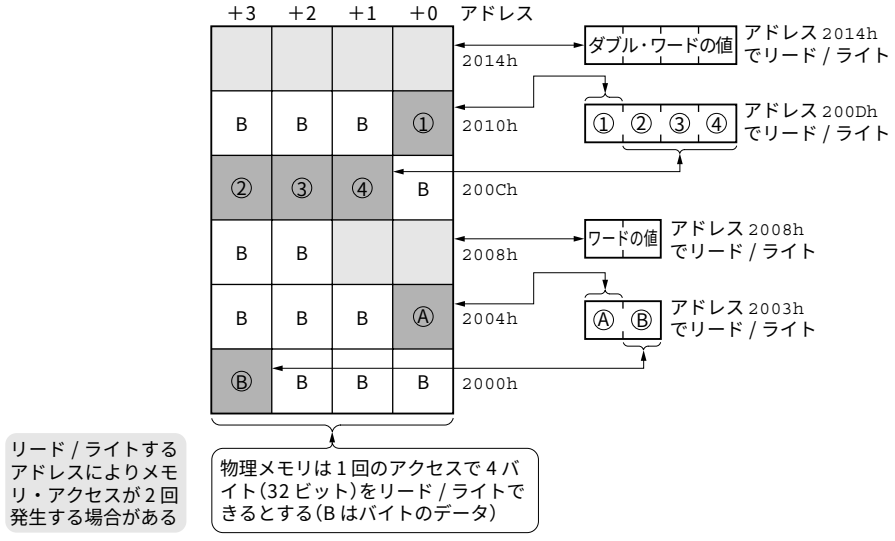

アプリケーション・プログラムからはOS(API)と共通ライブラリの呼び出しは可能だが，OS本体へのメモリ・アクセスはできない

**図4**
**アプリケーションが実行されるメモリ空間**




DW：ダブル・ワード(32ビット：4バイト)　W：ワード(16ビット：2バイト)　B：バイト(8ビット：1バイト)

**図5　メモリ上のバイト/ワード/ダブル・ワードの配置例**




リード / ライトするアドレスによりメモリ・アクセスが2回発生する場合がある

**図6**
**ワード/ダブル・ワードの悪い配置例**

データを定義するとき，アドレスがワードなら先頭バイトが2の倍数になるように，ダブル・ワードなら先頭バイトが4の倍数になるように配置することでこの問題を解決します(**図7**)．この作業を行うのが，MASMでは「ALIGNディレクティブ」，gasでは「.alignディレクティブ」となります．

▶MASMのALIGNディレクティブの記述のしかた

ALIGNディレクティブは，

```
ALIGN  [number]
```

と，ソース・ファイルに記述します(**リスト1**)．

ソース・ファイルに，ALIGNディレクティブを記述すると，次のデータや機械語命令をnumberの倍数

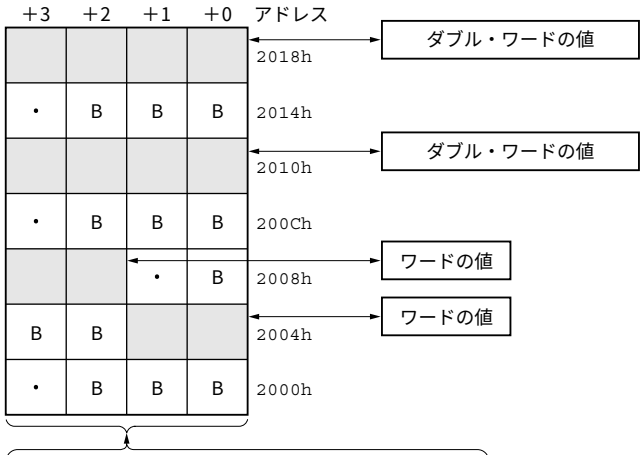



**図7**
**ワード/ダブル・ワードの良い配置例**

**リスト1**
**MASMのALIGNディレクティブ**

```
                                .586
                                .model flat

 00000000                       .data

                                        align   0
ap13_1.asm(6) : error A2063: can ALIGN only to power of 2 : 0
 00000000  01                           db      1

                                        align   1
 00000001  01                           db      1
                                        align   1
 00000002  01                           db      1

                                        align   2
 00000004  02                           db      2
                                        align   2
 00000006  02                           db      2

                                        align
 00000010  10                           db      16
                                        align
 00000020  10                           db      16

                                        align   4
 00000024  04                           db      4
                                        align   4
 00000028  04                           db      4

                                        align   8
 00000030  08                           db      8
                                        align   8
 00000038  08                           db      8

                                        align   3
ap13_1.asm(34) : error A2063: can ALIGN only to power of 2 : 3
 00000039  03                           db      3

                                end
```



に相当するアドレスから始まるようにします．numberを省略すると16の倍数のアドレスとなります．また，numberを指定する場合は，2の*n*乗で表される値を記述する必要があります．

▶gasの.alignディレクティブの記述のしかた

.alignディレクティブは，

```
.align  ABS-EXPR1,ABS-EXPR2,ABS-EXPR3
```

と，ソース・ファイルに記述します(**リスト2**)．

ABS-EXPR1，ABS-EXPR2，ABS-EXPR3は，ともに演算結果が絶対値の整数になる式です．ABS-EXPR1で，次のデータや機械語命令を配置するアドレスを指定します．するとアドレスはABS-EXPR1の倍数に相当するアドレスから始まるようになります．ABS-EXPR1では，2の*n*乗で表される値を記述する必

**リスト2**
**gasの.alignディレクティブ**

```
 1                      .data
 2                              .align  0
 3  0000 01                     .byte   1
 4                              .align  0
 5  0001 01                     .byte   1
 6
 7                              .align  1
 8  0002 01                     .byte   1
 9                              .align  1
10  0003 01                     .byte   1
11
12                              .align  2
13  0004 02                     .byte   2
14  0005 00                     .align  2
15  0006 02                     .byte   2
16
17                              .align
18  0007 10                     .byte   16
19                              .align
20  0008 10                     .byte   16
21
22  0009 111111                 .align  4,0x11
23  000c 04                     .byte   4
24  000d 121212                 .align  4,0x12
25  0010 04                     .byte   4
26
27  0011 00000000               .align  8
27       000000
28  0018 08                     .byte   8
29  0019 00000000               .align  8
29       000000
30  0020 08                     .byte   8
31
32  0021 00                     .byte   0
33  0022 20202020               .align  8,0x20,6
33       2020
34  0028 01                     .byte   1
35                              .align  8,0x20,6
36  0029 02                     .byte   2
```

gasでは0は1と同じ扱いになる

数字を指定しない場合は
.align 1
と同じ動作をする

詰め物が指定個数以下なら.alignを実行する

この場合.alignを実行すると詰め物が7個必要なため，アセンブラは.alignを実行しなかった

**リスト3**
**エラーとなる.alignディレクティブの使用例**

```
[hiro@osdsrv asm13]$ as -a ap13_3.s
ap13_3.s: Assembler messages:
ap13_3.s:5: Error: Alignment not a power of 2
ap13_3.s:7: Error: Alignment not a power of 2
ap13_3.s:9: Error: Alignment not a power of 2
GAS LISTING ap13_3.s                    page 1


   1                    .data
   2
   3  ???? 0000                   .align  4
   4  ???? 04                     .byte   4
   5                              .align  5
   6  ???? 05                     .byte   5
   7  ???? 00                     .align  6
****  Error:Alignment not a power of 2
   8  ???? 06                     .byte   6
   9                              .align  7
  10  ???? 07                     .byte   7
  11  ???? 000000                 .align  8
  12  ???? 08                     .byte   8
  13

GAS LISTING ap13_3.s                    page 2


NO DEFINED SYMBOLS

NO UNDEFINED SYMBOLS
[hiro@osdsrv asm13]$
```

.alignディレクティブは2″で表される値以外はエラーとなる．ただし，リスト上の表示ではエラーは一つしか表示していない？

要があります(**リスト3**)．

ABS-EXPR2は，現在のアドレスからABS-EXPR1の倍数のアドレスまでスキップするとき，その間に埋め込む詰め物の値を指定します．詰め物の値はバイト値です．ABS-EXPR2は省略可能で，省略した場合はゼロとなります．

ABS-EXPR3はオプションで，指定個数以下の詰め物が格納できるようなら.alignを実行します．

たとえば，

```
.align  8,0x20,6
```

の場合，詰め物が6個以下なら.alignを実行し，7個以上なら.alignは実行されません．

### 10.2.3　I/Oポートは物理的アドレスで指定

86系CPUは，メモリとは別にI/Oアドレス空間と呼ばれるI/Oポートを接続するための専用のアドレス空間をもっています．

WindowsやLinuxでは，メモリは仮想化されており，プログラムで使われるメモリ・アドレスも仮想的なものです．しかし，I/Oポートのアドレスは，物理的なアドレスが使われているため，プログラムがI/Oとして指定したアドレスは，そのままCPUにつながった物理的なI/Oアドレスとなります．

このI/Oアドレス空間は，バイト単位にアドレスが振られています．I/Oアドレス自体は16ビットあるため，64Kバイト分のI/Oが接続できます．

### 10.2.4　イミディエイトと実効アドレス

転送先のレジスタやメモリのサイズにあったイミディエイトの値を指定できます．整数データ転送で扱えるイミディエイトは，8ビット，16ビット，32ビットの三種類の定数です．

実効アドレスは，メモリにアクセスする際に使用するアドレスで，セグメントの先頭アドレスに加算される前のオフセットの値のことです．そのため，実効アドレスはつねに32ビット長の値となります．

## 10.3　データ転送命令のアセンブラ記述

今度は，整数データの転送に関する命令について説明します．データ転送命令は，32ビット命令のうち386CPU以降から，現在のPentium4まで共通して使用できるものについて先に説明します．

**表1**は，ここで説明する386CPU以降で使用できる32ビット転送命令を示したものです．

### 10.3.1　MOV命令

プログラム上，もっとも使用される命令はMOVで，

- 汎用レジスタ←汎用レジスタ
- 汎用レジスタ←メモリ
- メモリ←汎用レジスタ
- 汎用レジスタ←イミディエイト
- メモリ←イミディエイト
- セグメント・レジスタ←16ビット汎用レジスタ
- 16ビット汎用レジスタ←セグメント・レジスタ
- セグメント・レジスタ←ワード・メモリ
- ワード・メモリ←セグメント・レジスタ

のようなバリエーションがあります．

転送されるデータのサイズは，転送元，転送先ともに同じサイズ(8，16，32ビット長)です．

ただし，セグメント・レジスタに関する転送では，セグメント・レジスタが16ビット長なので，転送先あるいは転送元となる汎用レジスタやメモリは16ビット(ワード)のみとなります．

先にも述べたように，WindowsやLinuxで実行される32ビット・プログラムは，セグメント・レジスタの値をOSが固定しています．当然ですが，WindowsやLinux上で実行するアプリケーション・プログラムでは，セグメント・レジスタを転送先に指定することはできません．

実際のMASMでの記述例を**リスト4**に，gasでのMOVの記述例を**リスト5**に示します．

### 10.3.2　XCHG命令

XCHGは，オペランドで指定された二つの値を交換します．

交換できるのは，

- 汎用レジスタ⟷汎用レジスタ
- 汎用レジスタ⟷メモリ

で，交換できるデータのサイズは，両方ともに同じサイズ(8，16，32ビット長)となります(**リスト6**，**リスト7**)．

### 10.3.3　PUSH，POP命令

PUSHとPOPは，データをスタックに入れたり(プッシュ)出したり(ポップ)する命令で，16ビットおよび32ビットの値が扱えます(**リスト6**，**リスト7**)．

PUSHは，まずレジスタESPにこれから転送するデータのバイト・サイズだけマイナスします．次に指定された汎用レジスタやセグメント・レジスタ，メモリの値(16，32ビット)をSS：ESPが示すスタック領域へ転送します(**図8**)．

POPは，SS：ESPが示すスタック領域から，16ビットあるいは32ビットの値を読み出し，指定汎用レジスタやセグメント・レジスタ，メモリに転送します．転送後のレジスタESPは，転送したデータのバイト・サイズだけマイナスされます(**図8**)．

**表1**
**x86系の32ビットCPUで使用できる転送命令**

| インストラクション名 | 動　作 |
|---|---|
| MOV | Move<br>●汎用レジスタ間の転送<br>●汎用レジスタとメモリ間の転送<br>●汎用レジスタおよびメモリへのイミディエイトの転送<br>●セグメント・レジスタと汎用レジスタおよびメモリ間の転送 |
| XCHG | Exchange<br>●汎用レジスタ間の値の交換<br>●汎用レジスタとメモリ間の値の交換 |
| PUSH | ●スタックへの汎用レジスタ，セグメント・レジスタおよびメモリの値(16ビット，あるいは32ビット)を押し込む |
| POP | ●スタックから値(16ビット，あるいは32ビット)を引き上げ，汎用レジスタ，セグメント・レジスタおよびメモリにライトする |
| PUSHA<br>PUSHAD | Push all<br>●全汎用レジスタの値をスタックにPUSHする |
| POPA<br>POPAD | Pop all<br>●全汎用レジスタの値をスタックからPOPする |
| IN | Input<br>●I/Oポートからデータをリードする |
| OUT | Output<br>●I/Oポートへデータをライトする |
| CWD<br>CDQ | Convert Word to Double Word<br>Convert Doubleword to Quad Word<br>●ワード値をダブル・ワード値に変換する<br>●ダブル・ワード値をクワッド・ワード値に変換する |
| CBW<br>CWDE | Convert Byte to Word<br>Convert Word to Double Word in EAX register<br>●バイト値をワード値に変換する<br>●ワード値をダブル・ワード値に変換する |
| MOVSX | Move and sign extend<br>●符号拡張を伴う転送 |
| MOVZX | Move and zero extend<br>●ゼロ拡張を伴う転送 |
| LDS<br>LES<br>LFS<br>LGS<br>LSS | ●メモリ上のFARポインタをセグメント・レジスタと汎用レジスタにロードする |

注：表中の命令はすべてフラグを変化させない

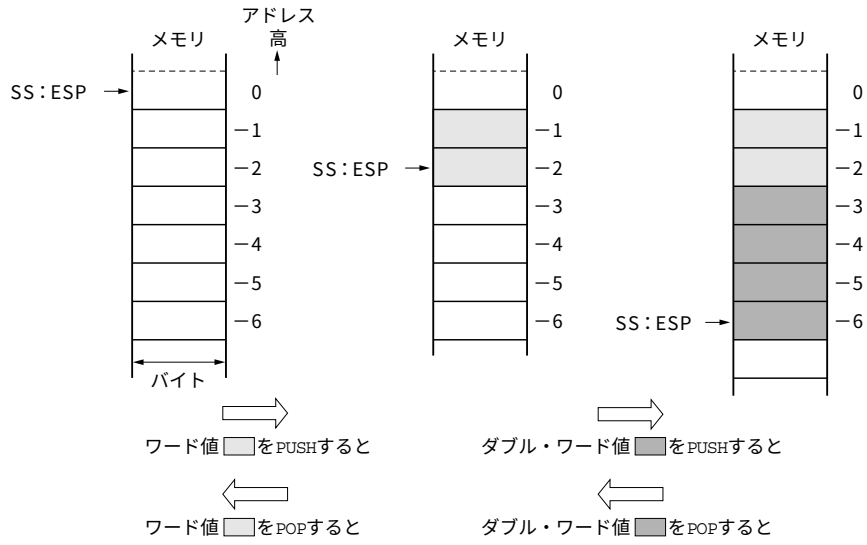



**図8**
**PUSH/POPの動作**

**リスト4**
**MASMのMOV命令の記述例**

```
                                .586
                                .model flat

00000000                        .data

00000000 01                     dtByte  db      1
00000001 0002                   dtWord  dw      2
00000003 00000004               dtDWord dd      4

00000000                        .code
                                ; 汎用レジスタ←汎用レジスタ
00000000  8A F8                         mov     bh,al
00000002  66| 8B DF                     mov     bx,di
00000005  8B F9                         mov     edi,ecx
                                ; 汎用レジスタ←メモリ
00000007  8A 1D 00000000 R              mov     bl,dtByte
0000000D  66| 8B 35                     mov     si,dtWord
          00000001 R
00000014  8B 0D 00000003 R              mov     ecx,dtDWord
                                ; メモリ←汎用レジスタ
0000001A  88 35 00000000 R              mov     dtByte,dh
00000020  66| 89 0D                     mov     dtWord,cx
          00000001 R
00000027  89 1D 00000003 R              mov     dtDWord,ebx
                                ; メモリ←アキュムレータ
0000002D  A2 00000000 R                 mov     dtByte,al
00000032  66| A3                        mov     dtWord,ax
          00000001 R
00000038  A3 00000003 R                 mov     dtDWord,eax
                                ; アキュムレータ←メモリ
0000003D  A0 00000000 R                 mov     al,dtByte
00000042  66| A1                        mov     ax,dtWord
          00000001 R
00000048  A1 00000003 R                 mov     eax,dtDWord
                                ; 汎用レジスタ←イミディエイト
0000004D  B7 12                         mov     bh,12h
0000004F  66| BB 1234                   mov     bx,1234h
00000053  BB 12345678                   mov     ebx,12345678h
                                ; メモリ←イミディエイト
00000058  C6 05 00000000 R              mov     dtByte,12h
          12
0000005F  66| C7 05                     mov     dtWord,1234h
          00000001 R
          1234
00000068  C7 05 00000003 R              mov     dtDWord,12345678h
          12345678
                                ; セグメント・レジスタ←16ビット汎用レジスタ
00000072  66| 8E C8                     mov     cs,ax
00000075  66| 8E DB                     mov     ds,bx
00000078  66| 8E C1                     mov     es,cx
0000007B  66| 8E E2                     mov     fs,dx
0000007E  66| 8E EE                     mov     gs,si
00000081  66| 8E D7                     mov     ss,di
                                ; 16ビット汎用レジスタ←セグメント・レジスタ
00000084  66| 8C C8                     mov     ax,cs
00000087  66| 8C DB                     mov     bx,ds
0000008A  66| 8C C1                     mov     cx,es
0000008D  66| 8C E2                     mov     dx,fs
00000090  66| 8C EE                     mov     si,gs
00000093  66| 8C D7                     mov     di,ss
                                ; セグメント・レジスタ←ワード・メモリ
00000096  66| 8E 0D                     mov     cs,dtWord
          00000001 R
0000009D  66| 8E 1D                     mov     ds,dtWord
          00000001 R
000000A4  66| 8E 05                     mov     es,dtWord
          00000001 R
000000AB  66| 8E 25                     mov     fs,dtWord
          00000001 R
000000B2  66| 8E 2D                     mov     gs,dtWord
          00000001 R
000000B9  66| 8E 15                     mov     ss,dtWord
          00000001 R
                                ; ワード・メモリ←セグメント・レジスタ
000000C0  66| 8C 0D                     mov     dtWord,cs
          00000001 R
000000C7  66| 8C 1D                     mov     dtWord,ds
          00000001 R
000000CE  66| 8C 05                     mov     dtWord,es
          00000001 R
000000D5  66| 8C 25                     mov     dtWord,fs
          00000001 R
000000DC  66| 8C 2D                     mov     dtWord,gs
          00000001 R
000000E3  66| 8C 15                     mov     dtWord,ss
          00000001 R

                                end
```

アキュムレータ（AL，AX，EAX）に対する直接アドレス（ディスプレースメントのみのアクセス）では，専用の機械語命令が用意されているため，ほかの汎用レジスタとメモリ間の転送に比べると機械語のバイト数が多少短くなる

Windowsで実行する32ビット・アプリケーションでは，セグメント・レジスタを変更することはできないため，この命令は使用しないこと

**リスト5　gasのMOV命令の記述例**

```
 1                      .data
 2
 3  0000 01             dtByte:   .byte  1
 4  0001 0200           dtWord:   .word  2
 5  0003 04000000       dtDWord:  .long  4
 6
 7                      .text
 8                      # 汎用レジスタ→汎用レジスタ
 9  0000 88C7                   movb    %al,%bh
10  0002 6689FB                 movw    %di,%bx
11  0005 89CF                   movl    %ecx,%edi
12                      # メモリ→汎用レジスタ
13  0007 8A1D0000               movb    dtByte,%bl
13       0000
14  000d 668B3501               movw    dtWord,%si
14       000000
15  0014 8B0D0300               movl    dtDWord,%ecx
15       0000
16                      # 汎用レジスタ→メモリ
17  001a 88350000               movb    %dh,dtByte
17       0000
18  0020 66890D01               movw    %cx,dtWord
18       000000
19  0027 891D0300               movl    %ebx,dtDWord
19       0000
20                      # アキュムレータ→メモリ
21  002d A2000000               movb    %al,dtByte
21       00
22  0032 66A30100               movw    %ax,dtWord
22       0000
23  0038 A3030000               movl    %eax,dtDWord
23       00
24                      # メモリ→アキュムレータ
25  003d A0000000               movb    dtByte,%al
25       00
26  0042 66A10100               movw    dtWord,%ax
26       0000
27  0048 A1030000               movl    dtDWord,%eax
27       00
28                      # イミディエイト→汎用レジスタ
29  004d B712                   movb    $0x12,%bh
30  004f 66BB3412               movw    $0x1234,%bx
31  0053 BB785634               movl    $0x12345678,%ebx
31       12
32                      # イミディエイト→メモリ
33  0058 C6050000               movb    $0x12,dtByte
33       000012
34  005f 66C70501               movw    $0x1234,dtWord
34       00000034
34       12
35  0068 C7050300               movl    $0x12345678,dtDWord
35       00007856
35       3412
36                      # 16ビット汎用レジスタ→セグメント・レジスタ
37  0072 8EC8                   movw    %ax,%cs
38  0074 8EDB                   movw    %bx,%ds
39  0076 8EC1                   movw    %cx,%es
40  0078 8EE2                   movw    %dx,%fs
41  007a 8EEE                   movw    %si,%gs
42  007c 8ED7                   movw    %di,%ss
43                      # セグメント・レジスタ→16ビット汎用レジスタ
44  007e 668CC8                 movw    %cs,%ax
45  0081 668CDB                 movw    %ds,%bx
46  0084 668CC1                 movw    %es,%cx
47  0087 668CE2                 movw    %fs,%dx
48  008a 668CEE                 movw    %gs,%si
49  008d 668CD7                 movw    %ss,%di
50                      # ワード・メモリ→セグメント・レジスタ
51  0090 8E0D0100               movw    dtWord,%cs
51       0000
52  0096 8E1D0100               movw    dtWord,%ds
52       0000
53  009c 8E050100               movw    dtWord,%es
53       0000
54  00a2 8E250100               movw    dtWord,%fs
54       0000
55  00a8 8E2D0100               movw    dtWord,%gs
55       0000
56  00ae 8E150100               movw    dtWord,%ss
56       0000
57                      # セグメント・レジスタ→ワード・メモリ
58  00b4 668C0D01               movw    %cs,dtWord
58       000000
59  00bb 668C1D01               movw    %ds,dtWord
59       000000
60  00c2 668C0501               movw    %es,dtWord
60       000000
61  00c9 668C2501               movw    %fs,dtWord
61       000000
62  00d0 668C2D01               movw    %gs,dtWord
62       000000
63  00d7 668C1501               movw    %ss,dtWord
63       000000
64
```

Linuxで実行するアプリケーションでは，セグメントを変更することができないためこの命令は使用しないようにする．また，ここで使用しているgasは，16ビット・オペランドを使用するためのオペランド・プリフィックスが出力されていないので使用できない