アセンブラを理解するための必須知識

第1章 メモリの概念を理解する

中森 章

アセンブラを知るためには，それが動くCPUを知らなければならない．逆に言うと，CPUの動作さえ把握すれば，CPUが解釈できるように，アセンブラでプログラムを記述することで，CPUはそのとおりに動作する．

そこで本章では，まずCPUのアーキテクチャに焦点を当てて解説し，それに付随してメモリの概念について基本的なところから解説する．また，メモリへのアクセスが実際にどのようにして行われるかを図解することにより，CPUの動きを理解する．（編集部）

なぜいまアセンブラなのか

● アセンブラはCPUの機能を最大限に引き出せる言語

いわゆる高級言語全盛の時代に，なぜアセンブラの知識が必要なのかと思われるかもしれません．しかし，アセンブラを知るということは使用しているCPUの機能を最大限に引き出せる機会を得るということでもあります．

▶ 高速に実行させるためにアセンブラを使う

アセンブラはCPUを直接制御できる言語です．その意味で，プログラムの速度を極限まで追及する場合にはアセンブラを使うことになります．しかし，これは少し昔の認識かもしれません．各CPUメーカがコンパイラの最適化に多額の費用をつぎ込むようになった昨今では，コンパイラは十分に賢くなっています．ときとして，コンパイラは人間が書いたものよりも高速実行を行う命令列を生成します．それでもアセンブラを使うということはCPUを最適に扱えているという幻想かもしれません．

しかし，そこにロマンを感じる人も多いと思います．コンパイラが生成するよりも高速な命令列を記述できた暁には，不思議な達成感を得ることができるのも確かです．

▶ アセンブラでしか実現できない処理がある

また，コンパイラがサポートしておらず，アセンブラでないと実現できない処理もあります．たとえばビット操作です．C言語は高級言語の中でもビット操作をサポートしている特異な言語であるといえます．もっとも，C言語は歴史的にはミニコン（PDP-11）のアセンブラを簡単に使えるようにしたものという説がありますから，アセンブラと同等といっても過言ではないのですが…．

話が横道に逸れましたが，コンパイラは基本的にはユーザが使用するアプリケーション・プログラムをコンパイルすることを目的としています．その意味で，割り込み処理や特権命令の使用などOSの核となる部分はアセンブラで記述するしかありません．

あるいは，特定のCPU（DSPを含む）のために拡張された命令セット，コプロセッサとのインターフェースなど，コンパイラでは一般的にサポートできない命令を記述するためにもアセンブラが必要です．たとえば，乗算と加算を1命令で行う積和演算命令をもつDSPがありますが，このような命令をサポートしているコンパイラはあまり見たことがありません．これらの拡張命令を使えば，当然ながら性能の向上が期待できるので，アセンブラが性能向上の手段といってもよいかもしれません．

● アセンブラとCPUの基礎を身につけよう

本章の第一の目的は，コンパイラが生成するアセンブラの基礎を知ることにあります．たとえば，ポインタの概念はアセンブラで身につけると，高級言語でのポインタの使い方に対する理解が深まると思います．

第二の目的は，コンパイラの生成する命令列を手作業で最適化するための勘所を与えることにあります．

具体的な個別のCPUのアセンブラの例は別の章で扱うので，ここでは仮想的なCPUを定義して，アセンブラの特徴と使用方法を見ていきます．

命令セットはCPUを映す鏡

● アセンブラ，アセンブリ言語，アセンブラ言語

アセンブラ，アセンブラ言語と，とくに説明なく使用してきましたが，ここでことばの定義をしておきましょう．

まず，CPUを直接制御する命令は，昔は（今でも）機械語あるいはマシン語と呼ぶことがありました．CPUが直接読み込んで理解できるのは0と1のビット列の塊でしかありません．CPUは，これを8ビット，16ビット，あるいは32ビット単位のデータとして読み込んで，命令であると解釈します．

機械語は命令を2進数あるいは16進数で表した実にハナモゲラ（死語？）な言語でした．そのため，その機械語を覚えやすい

KEYWORD ―アセンブラ，CPU，メモリ，エンディアン，コンパイラ，命令セット，ビット操作，機械語

ようにADDとかSUBとかの記号を一対一に対応させて使っていました(さすがに英語が基になっているが…)．この記号のことをニモニック(mnemonic＝記憶術)といいます．このニモニックを使って機械語を記述したものがアセンブリ(assembly＝組み立て)言語です．アセンブリ言語を，CPUが理解可能な，0や1のビット列にアセンブル(assemble＝組み立てる，変換する)するのがアセンブラ(assembler＝アセンブルするもの)です．アセンブリとはアセンブルの名詞形です．

本文中では，アセンブリ言語を意図的にアセンブラ言語と表記している部分があります．これは，アセンブラに入力する言語をJIS(日本工業規格)ではアセンブラ言語と定義していることもありますが，アセンブラのための言語という意味を込めています．また，高級言語をアセンブリ言語にコンパイル(compile＝集める，編集する)するコンパイラ(compiler＝コンパイルするもの)を考える場合，高級言語のことをコンパイラ言語と呼ぶことはありますが，コンパイルの名詞形のコンピレーション(compilation)言語ということはほとんどないという理由もあります．ただ，コンピュータ発祥の地である英国や米国では，アセンブリ言語(assembly language)と表現するのが弱いところですが…．

本章では，主として，アセンブラ，アセンブラ言語という表現を使いますが，二つを厳密には区別していません．実生活でも区別せずに使う場合もあります．とくに断らない限りは状況に応じて判断してください．

● CPU設計者の思いは命令セットに現われる

アセンブラとCPUの間には切っても切れない関係があります．あるCPUがどのような機械語の命令を有しているかを「命令セット(命令の集合)」といいます．また，命令セットを含めて，「アドレッシング・モード(データの格納場所を指し示す方式)」にはどのようなものがあるか，「レジスタ」(後で説明)は何本あるかといった，プログラマに見えるCPUの特徴を命令セット・アーキテクチャといいます．

世の中には種々のCPUが存在しています．それらCPUが有している命令セット・アーキテクチャも千差万別です．命令セット・アーキテクチャには設計者の思想が込められているのが一般的です．つまり，機械語命令はCPUを直接操作できる唯一の手段であり，その機械語命令を組み合わせて実現できることが，そのCPUのできるすべてのことだからです．CPUの命令数は，通常は100から200種程度，多くても400種程度です．このどちらかといえば少数の命令の組み合わせが，ロケットを飛ばしたり人間のことばを理解したりチェスの王者を作り上げたりするのです．古いたとえでいえば，鉄人28号がリモコンのたった2本のレバーの組み合わせで敵ロボットと複雑な格闘をするようなものでしょうか．

閑話休題．たとえば，CISCアーキテクチャの集大成ともいえるTRONチップを思い出してみましょう．1986年ごろにTRONチップの提唱者である坂村健氏は，その命令セットの特色を至る場所で熱く語っていました．その本質は，ソフトウェア危機を見込み，人類の大多数がプログラマ化する時代に備えて，コンパイラに優しい(コンパイラを作りやすい)命令セットを提供することでした．これは，コンパイラ言語(主としてC言語)とアセンブラの類似性を体感するという本章の目的と似ています．そのため，ここでは，RISC全盛の現代において，あえてCISCに似た命令セットを仮想CPUの命令セットとして後ほど提案します．

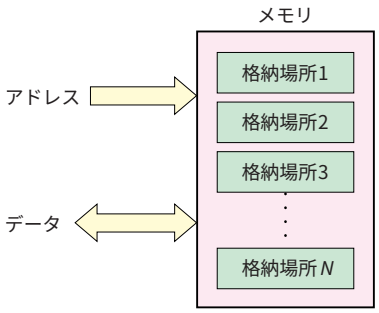

図1 メモリの概念(その1)
アドレスを与えると格納場所が一意に定まり，その格納場所とデータをやり取り可能な装置

メモリの概念とプログラムの作成

● はじめにメモリありき

CPUはメモリに置かれたプログラム(機械語の命令列)を実行します．プログラムが扱うデータもまたメモリに置かれます．逆にいえば，メモリがないとCPUはプログラムを実行することはできません．ここでは，プログラムの要となる，メモリについて考えてみましょう．

● メモリ空間(アドレス空間)

メモリとは何でしょう．ROMやRAMと呼ばれるICチップを思い浮かべる人も多いと思います．しかし，ROMやRAMに限らず，アドレスと，そのアドレスに一対一に対応する格納領域が定まり，アドレスを拠り所にその格納領域との間でデータをやり取りする記憶装置全般をメモリといいます(**図1**)．

メモリ・ライトとは，アドレスとデータを与えて，アドレスに対応する格納場所にデータを詰め込むこと，メモリ・リードとは，アドレスを与えて，アドレスに対応する格納場所からデータを取り出すことをいいます．メモリ・ライトとメモリ・リードの総称をメモリ・アクセスといいます(**図2**)．

このメモリを書き換えるのは，基本的に，CPUのみです．メモリはCPUを使って意識的に書き換えない限り，勝手に書き換えられることはありません(この説明は，DMAやマルチプロセッサ環境ではかならずしも正しくないが，とりあえず無視する)．そして，そのCPUにメモリをアクセスする指示を行うのはプログラムなのです．

いわゆる32ビットCPUにおいて，メモリを指し示すための

Information ―― MSNサーチの日本語版および中国語版向けソフトウェアに「Rosette&reg」が採用
MSNの(http://www.msn.co.jp/)検索サービスに，米Basis Technology社の日本語および中国語の形態素解析システム「Rosette&reg」が採用された．同ソフトウェアは，多言語で書かれたクエリの理解や，Webページの検索を行う機能を提供するもの．

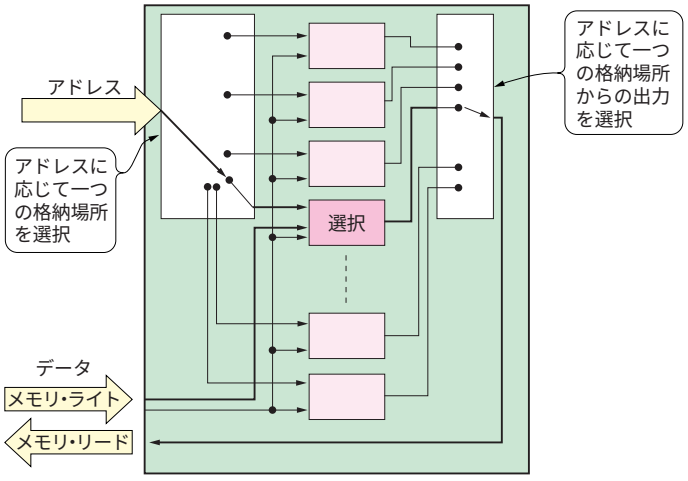

図2　メモリ・アクセス――アドレスの指定によりメモリの内容を書き込み/読み出しする

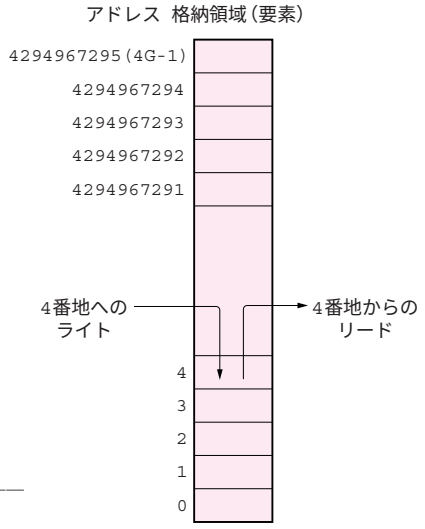

図3
メモリの概念（その2）――
4G種類の配列がある

アドレスは，通常，32ビットの整数です．このため，

0x00000000 ～ 0xFFFFFFFF

の4294967296種類の値をとることができます．これはG（ギガ＝2^{30}＝1073741824）という単位を使えば4G種類です．つまり，メモリの格納場所には0から（4G－1）までの番地が付いていることになります．というか，「アドレス（address）」を日本語に訳せば「番地」ということですね．ちなみに，一般人の世界では1000を単位として数値を表しますが，コンピュータの世界では1024（＝2^{10}）を単位とします．Gのほかには，M（メガ＝2^{20}＝1048576），K（キロ＝2^{10}＝1024），T（テラ＝2^{40}），P（ペタ＝2^{50}），E（エクサ＝2^{60}）などがあります．

一般的なCPUではデータの格納場所の大きさは8ビット（＝1バイト）なので，メモリは1バイトごとに番地をもっていることになります．つまり，この場合，CPUがアクセスできるメモリは4Gバイトの大きさをもつことになります．このメモリの広がりのことをメモリ空間と呼びます．以上の知識から，たとえば「アドレス0x00001000にはデータ0xffが入っている」，「アドレス0xffff0000にデータ0xaaを書き込む」というように使います．

話が飛びますが，配列というデータ構造を知っている人なら，メモリ空間とは0から（4G－1）の添え字をもつ1バイト・データの1次元配列として認識できると思います（**図3**）．

● 32ビットのデータを複数のメモリに格納する

32ビットCPUにおいて，CPUが扱うもっとも自然なデータは32ビット（長の）整数です．これは4バイトの大きさなので，32ビット整数をメモリに格納する場合，メモリ空間の中では四つの格納場所を必要とします．つまり，アドレスとしては

a，（a＋1），（a＋2），（a＋3）

の四つの値をとります．このとき，この32ビット整数は「a番地に格納されている」と，いちばん小さなアドレスの値に代表させて格納位置を示します．

CPUが扱うデータとしては，32ビット整数のほかにも，16ビット整数，8ビット整数があります．32ビット整数の場合と同様に，16ビット整数は二つのアドレスで2バイトの格納場所を，8ビット整数は一つのアドレスで1バイトの格納場所を占有します．これらの場合も，各データが存在する位置はいちばん小さなアドレスで示します．

つまり，CPUに「a番地から32ビット・データをもって来い」と命令する場合，CPUの動作としては，a，（a＋1），（a＋2），（a＋3）番地から合計4バイトのデータを取り出して32ビットに結合して取り出すことになります．「a番地から16ビット・データをもって来い」と命令する場合，a，（a＋1）番地から合計2バイトのデータを取り出して16ビットに結合して取り出します．「a番地から8ビット・データをもって来い」と命令する場合，a番地のデータをそのまま取り出します．

ここで，勘の良い人なら気づくと思いますが，メモリに8ビットごとに番地が付いている理由は，最小で8ビットのデータを扱う必要があるからです．つまり，8ビットごとに番地を付けておけば，8ビットの倍数である，16ビット，（場合によっては）24ビット，32ビットのデータを扱うことも可能になるのです．逆にいえば，CPUが32ビット・データしか扱わないのであれば，何も8ビットごとにメモリの番地を付ける必要はなく，32ビットごとに番地を付ければ十分です．

● メモリにデータを格納する順番（エンディアン）

ところで，32ビットのデータを4個の1バイトの格納場所に格納する場合，あるいは，16ビットのデータを2個の1バイトの格納場所に格納する場合，データをどのような順序で格納するかが問題になります．単純に考えれば，データを8ビットごとに区切って順番に格納すればよいのですが，この「順番」というのが曲者（くせもの）です．たとえば，

0x12345678

という32ビットの整数を考えましょう（0xは16進数を表すと

News Flash――AMDとIBM，65nmプロセス技術を発表
米AMD社と米IBM社は，米国で開催されたInternational Electron Devices Meetingにおいて，従来のトランジスタよりも性能が約40％向上した65nmプロセス技術を発表した．

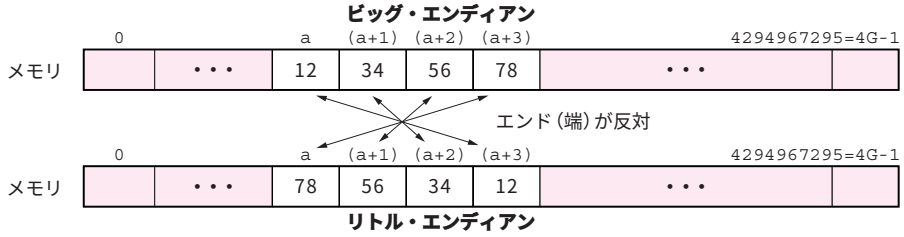

図4
エンディアン
(0x12345678の格納のされ方)

いう表記法)．素直に8ビットごとに区切れば，

```
0x12，0x34，0x56，0x78
```

となります．さて，これをどのような順番でメモリに格納すればよいでしょうか．ある人は，

(a＋0)番地 ← 0x12
(a＋1)番地 ← 0x34
(a＋2)番地 ← 0x56
(a＋3)番地 ← 0x78

とするかもしれません．またある人は，

(a＋0)番地 ← 0x78
(a＋1)番地 ← 0x56
(a＋2)番地 ← 0x34
(a＋3)番地 ← 0x12

とするかもしれません．さすがに，

(a＋0)番地 ← 0x56
(a＋1)番地 ← 0x12
(a＋2)番地 ← 0x78
(a＋3)番地 ← 0x12

などと，8ビット単位の並びをバラバラに格納するひねくれ者はいないと思います．常識的には，1番目のように，位の1番高いデータを1番低いアドレスに格納する方法と，2番目のように，位の1番低いデータを1番低いアドレスに格納する方法の2種類です．

▶ビッグ・エンディアンとリトル・エンディアンはIntelとMotorolaの対立のなごり

データをどちらの順番で格納するかは，好みというか主義の問題なので，一概にどちらが優れているかという判断はできません．実際に，どちらの方法を採用するかはCPUメーカによってまちまちです．前者はビッグ・エンディアン(big endian)，後者はリトル・エンディアン(little endian)と呼ばれています．もっとも自然なデータ長である32ビットのメモリ領域(連続する4バイト)を考える場合，いちばん低いアドレスがビッグ・エンディアンとリトル・エンディアンで正反対の端になります(**図4**)．

マイクロプロセッサでいえば，昔は二大勢力だった，Intel(x86)とMotorolaが逆のエンディアンを採用しているのが特徴的でした．Intelはリトル・エンディアン，Motorolaはビッグ・エンディアンです．ワークステーションのCPUでは，DECがリトル・エンディアン，IBMがビッグ・エンディアンでした．

昔，まだエンディアンという単語が一般的でない時代は，リトル・エンディアンをIntel形式，ビッグ・エンディアンをMotorola形式と呼んでいました．

しかし，一つのシステムの中でさまざまなCPUが混在する現状においては，そのシステムに採用されているCPUが同一のエンディアンであるとは限りません．ビッグ・エンディアンのシステムで生成されたデータをリトル・エンディアンのシステムで受け取るという状況も珍しくありません．このような状況に対応するため，とくに組み込み制御用のCPUは，ビッグ・エンディアンで動作するかリトル・エンディアンで動作するかを起動時に選択できるようになっています．あるいは，動的にエンディアンを切り替えながら実行するCPUもあります．

このようにビッグ・エンディアンにもリトル・エンディアンにも対応可能な性質をバイ・エンディアン(bi-endian)といいます．最近のSuperH，ARM，MIPS，PowerPCといったCPUはすべてバイ・エンディアンです．x86だけは，かたくなに，リトル・エンディアンを維持していますが…．

▶32ビット・データでは同一のデータが見える

ときどき，エンディアンを整数データそのもののバイト並び，あるいはビット並びと思っている人がいますが，これはまちがいです．エンディアンとはデータの「(バイト・アドレスで構成される)メモリ空間での」バイト並びであり，メモリが絡まない場合は意味をなしません．また，メモリとCPUが32ビット・バスで接続される場合には，CPUから見えるデータとしては同一のビット・イメージになります(**図5**)．これが，64ビット・バスで接続される場合は，32ビット単位で上下逆転します．16ビット・バスでは，ビッグ・エンディアンでは上位16ビット(0x1234)が，リトル・エンディアンでは下位16ビット(0x5678)が見えることになります．つまり，32ビット長のデータ・アクセスでは，ビッグ・エンディアンでもリトル・エンディアンでも，同一のデータが見えることになります．その意味で，32ビット・データはエンディアンにとって特別なデータ長といえます．

このように，リトル・エンディアンでは，アドレスの高い方向に上位のデータが来るので，あるアドレスから8ビット，16ビット，32ビットとデータ長を変えてアクセスした場合は，自然に拡張されるように見えます．一方，ビッグ・エンディアンでは，データ長を変えてアクセスした場合は，データ長に応じ

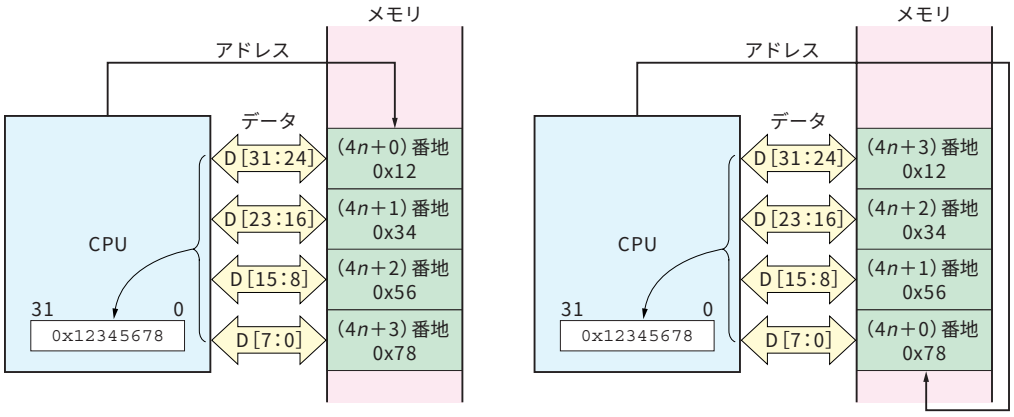

図5
エンディアンとデータ・バスの結線
CPU側から見れば，どちらの場合も，データ・バスに0x12345678が乗っているように見える．CPUが32ビット・データとして取り込む場合も同じ

リスト1　エンディアン判別プログラム（その1）

```
union {
   char b[4];
   int  w;
} t;

t.b[0]=0x11;
t.b[1]=0x22;
t.b[2]=0x33;
t.b[3]=0x44;
if(t.w==0x44332211)
   printf("Little Endian¥n");
else if(t.w==0x11223344)
   printf("Big Endian¥n");
else
   printf("Impossible!¥n");
```

リスト2　エンディアン判別プログラム（その2）

```
union {
   char b[2];
   short h;
} t;

t.b[0]=0x11;
t.b[1]=0x22;
if(t.h==0x2211)
   printf("Little Endian¥n");
else if(t.h==0x1122)
   printf("Big Endian¥n");
else
   printf("Impossible!¥n");
```

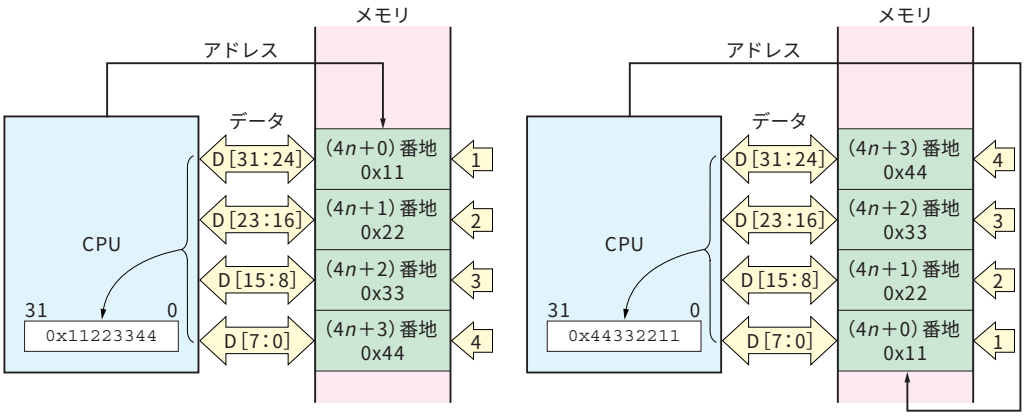

図6
エンディアンの特徴
あるアドレスから8ビット単位に0x11，0x22，0x33，0x44を書き込んで，32ビットでリードすると，エンディアンの違いでCPUに取り込まれるデータが異なる

て（バイト単位に）データが逆転してみえます．この特徴を利用すれば，現在使用しているコンピュータがリトル・エンディアンかビッグ・エンディアンかを判別することができます．つまり，**リスト1**や**リスト2**のようなプログラムを書けばよいのです．**リスト1**のプログラムを図示すると**図6**のようになります．

●「ワード」ということばと諸注意 ――方言？

なお，CPUが扱うもっとも自然なビット長のデータをワードといいます．その意味で，32ビットCPUでは，32ビット・データをワード，16ビット・データをハーフ・ワードと呼びます．8ビット・データだけはクォータ・ワードではなくバイトと呼ぶのが普通です．

一方，16ビットCPUでは，16ビット・データをワード，32ビット・データをロング・ワード，またはダブル・ワードと呼びます．ところが，x86系のCPUでは，CPU自体の構造は32ビットになっていますが，16ビット時代からの継承性のためか，いまだに16ビット・データをワードと呼んでいます．x86

News Flash ―― Intel と STMicroelectronics，携帯電話向けにフラッシュ・メモリ・サブシステムを共同で開発
米Intel社とスイスのSTMicroelectronics社は，携帯電話端末のメモリ部の設計を簡素化できる共通仕様を作成し，それに基いたハードウェアおよびソフトウェア互換のあるフラッシュ・メモリ製品を提供すると発表した．