

[機械語序論 2回目 2002・12・10]

今回はアセンブリ言語のプログラムの書き方と x86 のもっとも基本的な命令の使い方について説明します。

アセンブラ基本的な記法

前回、機械語とは何かということについて説明しましたが、機械語に 1 対 1 に記述するのがアセンブリ言語です。アセンブリ言語は、基本的に以下のような形式で記述します。

命令コードのニーモニック オペランド 1、オペランド 2、オペランド 3、...

オペランドの数や使えるオペランドの種類については、命令コードによって異なります。

x86 のアセンブリ記法については、AT&T 記法と Intel 記法がありますが、Linux で使えるアセンブラは gnu のアセンブラ gas で、これは AT&T 記法を使っていますので、ここでは AT&T 記法について説明していきます。

x86 アセンブリ言語のオペランドの種類と記法 (その 1)

オペランドには、以下の種類があります。

- **レジスタ** : x86 には 32 ビットの汎用レジスタとして、eax, ebx, ecx, edx, esi, edi, esp, ebp の 8 個のレジスタがあります。このうち、esp はスタックポインター、ebp はベースレジスタと呼ばれているもので使い方が決まっていますので (これは、関数呼び出しで説明する) 当面使えるレジスタは 6 個です。オペランドにレジスタを指定する場合には、レジスタ名の前に % をつけます。例えば、eax をオペランドに指定する場合には、%eax と書きます。
- **即値 (イミディエイト: immediate)** : これは、定数のことです。定数をオペランドに指定する場合には、最初に \$ をつけます。例えば、1 をオペランドに指定する場合には、\$1 と書きます。これは、10 進数です。16 進数を指定する場合には、0x をつけます。また、\$ のあとには +、- などの簡単な式をかくことができます。
- **アドレス参照** : \$ をつけずに、単なる数値をオペランドとして書いた場合はその数値で示されるアドレスに対するメモリ参照になります。例えば、100 と書いた場合にはアドレス 100 を参照します。また、データ領域につけた変数のラベル (名前) を書いた場合にも、そのラベルのアドレスを参照します。

このほかにも、レジスタによるメモリ参照 (アドレッシングモードと呼ばれる) や ax や al, ah などレジスタの部分指定がありますが、これについては、(その 2) で解説することにします。

mov 命令

もっとも基本的な命令は mov 命令です。

```
mov src, dst
```

src で指定されたオペランドから、dst で指定されたオペランドにコピーします。

```
mov $1,%eax # レジスタ eax に 1 をセット
mov %eax,%edi # eax の内容を edi にコピー
mov 100,%ebx # アドレス 100 の 32 ビットワードを ebx にロード
mov %edx, 200 # アドレス 200 に、edx の内容をストア
mov x,%ecx # ラベル x のデータを ecx にロード
```

上説明した大抵の組み合わせはつかえますが、以下の制限があります。

src と dst の両方がアドレス参照であってはならない

ということです。例えば、mov 100,200 はエラーになります。

add 命令、sub 命令

加算を行う add 命令と、減算を行う sub 命令はもっとも基本的な演算命令です。

```
add src, dst # dst = dst + src
sub src, dst # dst = dst - src
```

x86 の命令は dst について、加算、減算を行う 2 オペランドの命令であることを注意してください。

```
add $1,%eax # レジスタ eax に 1 を加算する。
sub 100,%edx # レジスタ edx から、アドレス 100 の内容を減算する
add %edx,%ebx # レジスタ ebx に edx の内容を加算する。
sub %eax,x # ラベル x のデータから、eax の値を減算する。
```

ここでも、src と dst の両方がアドレスであってはなりません。

cc -S で得られたコードで、movl とか addl など、l がついている場合があります。これは、命令が扱

データのサイズをあらわすもので、lは32ビット、wは16ビット、bは8ビットをあらわします。たとえば、movlは32ビットのデータのmov命令、movwは16ビットのデータのmov命令です。オペランドのどれかがレジスタでeaxなどの32ビットのレジスタの場合はデータのサイズがわかりますので、mov命令をつかっても自動的にmovlと同じと解釈されます。

アセンブラ擬似命令 (その1)

アセンブリ言語には、命令をあらわす部分のほか、記法上便宜的に導入された擬似命令があります。

- **ラベル**： 名前のあとに:をつけたものはラベルで、プログラム中の位置やアドレスを示します。これは、C言語のラベルと同じです。
- **.text**： プログラムのコードであることを示します。これによって、コードはまとめられて、メモリ上に格納されます。
- **.data**： プログラムのデータ部分であることをしめします。データもデータだけまとめられて、メモリ中に格納されます。
- **.globl ラベル**： ラベルをリンク時に見えるようにします。他のファイルから参照される名前は.globlで宣言しておかなくてはなりません。
- **.word n**： 16ビットのnという値を格納する領域を確保します。
- **.long n**： 32ビットのnという値を格納する領域を確保します。
- **.align 4**： 4バイトごとの境界にあわせませす。

このほかにもありますが、それはこれからの講義で説明することにするにすることに、右のはxという32ビットの領域を確保して、その領域にラベルxをつけておきます。xに1を加えるプログラムの1部です。C言語と同じようにmainから始まるものとして、mainは、.globlで外から参照できるようにしておきます。.alignはなくても動きますが、4バイト境界にないと、効率が下がりますので、つけておきます。

あと、アセンブリ言語でのコメントは#で始めるか、Cと同じように/* */で囲った部分がコメントになります。

```
.data
.align 4
x: .long 100
.text
.align 4
.globl main
main: mov x,%eax
      add $1,%eax
      mov %eax, x
      ...
```

簡単な分岐命令 (その1)

では、この時点で書けるプログラムを面白くするために最後に簡単な分岐命令を説明しましょう。C言語でifに相当する分岐命令は、cmp命令と条件分岐命令の組み合わせでかきます。

```
cmp opd1, opd2
```

```
je L
```

je命令は、前のcmp命令で、比較した結果が同じであればコードにあるラベルLに分岐する命令です。例えば、

```
cmp %eax,$0
```

```
je L
```

では、eaxが0だったら、ラベルLに分岐します。ちなみに、同じでなければ分岐する命令はjneです。

また、無条件に分岐する命令は、jmp命令です。

```
jmp L
```

これは、C言語でいえばgoto文で、コード中のラベルはC言語と同じように、プログラム中にL:という形式で書いておきます。

右の例は、eaxを0にしておいて、eaxを1ずつ加えて、eaxが10になるまで、ループする例です。

```
mov $0, %eax
L1: cmp $10,%eax
    je L2
    ... # ループ本体
    add $1,%eax
    jmp L1
L2:  ....
```

今回やったことのまとめ：

オペランド (レジスタ、即値、アドレス)、mov命令、add/sub命令、アセンブラ擬似命令 (.text, .data, .globl, .align, .long, .word)、cmp命令、je/jne命令、jmp命令