

[機械語序論 2回目]

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ことにして、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 $0,%eax
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。なお、cmp 命令は sub 命令と同じで被減算数が右に来ることになっていますので、イミディエトの値 (\$10 等) は左のオペランドに書かなくてははいけません。

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

演習提出について

- 演習課題の提出は、電子メールにて、
kikaigo@hpcs.cs.tsukuba.ac.jp
あてに提出すること。
- 必ず Subject に “kikaigo-kadai 課題番号” と記すこと（例：“kikaigo-kadai 1” 等）。この指示に従っていない場合、レポートのメールとみなされず無視する場合があるので注意すること。
- 内容の冒頭に、必ず学籍番号と氏名を記述すること。
- 提出の締め切りは、次回の演習の終了時までとする。
- 電子メールで送る場合は、原則として coins の計算機で行うこと。

課題プログラムの作成について

特に指定のない場合には、以下のようにして、課題のアセンブラプログラムを作成する。

- 作成するプログラムのファイル名は、.s で終わるものにする。例えば、test.s など。以降、ファイル名を test.s として説明する。
- プログラムの始まりは main とする。
- プログラムの最後で、call stop として終わること。stop は、各レジスタの内容をプリントアウトしてプログラムを終了するルーチンで、/home/lab/Denjo/msato/libkikaigo.a にある。
- すなわち、test.s は、以下ようになる。

```
.text
.align 4
.globl main
main: /* ここからプログラムを書く */
    ...
    call stop # これでプログラム終了
```

注意：データの宣言等は、この前でもよい。

- アセンブル（コンパイル）は、cc コマンドで行う。
まず、/home/lab/Denjo/msato/libkikaigo.a を自分のディレクトリにコピーする。
cc コマンドでアセンブル、リンク
% cc test.s libkikaigo.a
できた実行ファイルを実行する
% a.out
ここで、レジスタの内容をプリントアウトして終了することを確認

/home/lab/Denjo/msato/sample.s は、eax に 1、ebx を 2 にセットして、加算した結果を ecx にセットして終了するプログラムです。これを実行して、プリントアウトの結果をみてください。

課題 1

- 1、eax, ebx, ecx, edx のそれぞれのレジスタに 1, 2, 3, 4 の値をセットし、それらを esi に加算して、終了 (call stop) するプログラムを書きなさい。プリントアウトされるレジスタの内容を見て、esi に 10 がセットされていることを確認すること。
- 2、講義の最後で説明したループのコードを参考にして、1 から 10 までの値の加算をして、その結果を ebx にセットして終わるプログラムを書きなさい。プリントアウトされるレジスタの内容をみて、55 がセットされていることを確認すること。

アセンブラプログラムのデバックの方法

アセンブラプログラムのデバックは、gdb (gnu debugger) を使って行うことができます。

gdb の起動

gdb は、単独でも起動することができますが、emacs から起動すると便利です。実行プログラムを a.out とすると、まず、emacs から、

```
M-x gdb
```

と入力します。Run gdb (like this): gdb とプロンプトでるので、ここで、a.out と入力し、リターンします。そこで、gdb の window が開かれるはずですが。

ブレークポイントの設定と実行開始

課題のプログラムは main から始まるので、まず、ここで停止するように、break コマンドで main にブレークポイントを設定します。(gdb) とプロンプトがあるので、ここで、

```
(gdb) break main
```

と入力します。次に、run コマンド main まで実行します。

```
(gdb) run
```

すると、実行が始まり、main で停止するはずですが。

プログラムの disassemble

ここで、プログラムがどのようなコードになっているかについて、確認してみましょう。メモリ上の機械語になったプログラムをアセンブリプログラムで表示するのが disassemble コマンドです。disassemble とは、アセンブルの反対、つまり、機械語からアセンブラに直すことです。main から始まるプログラムを disassemble してみましょう。

```
(gdb) disassemble main
```

main のところに、任意のラベル名を書くことでそのプログラムを disassemble することができます。

プログラムのステップ実行

1 命令ずつ実行するコマンドが、stepi です。

```
(gdb) stepi
```

ここで、stepi コマンドを実行するごとに 1 命令ずつ実行されているのがわかるはずですが。

レジスタの表示

step 実行している途中で、レジスタの表示をして見ましょう。表示には 2 つの方法があります。

```
(gdb) info registers
```

では、すべてのレジスタの表示を行います。個別のレジスタを表示する場合には、

```
(gdb) print $レジスタ名
```

で表示させることができます。

実行の再開、ブレークポイントの設定

continue コマンドは実行を次のブレークポイントまで (もしくは終わりまで) 実行を再開するコマンドです。

```
(gdb) continue
```

さて、main にブレークポイントを設定しましたが、main の代わりにラベル名を書くことで、そのラベルの前で実行を止めることができます。また、アドレスを指定したい場合には

```
(gdb) break *アドレス
```

で任意のアドレスで実行を中断することができます。

データの表示

データの表示を行うコマンドが x コマンドです。

```
(gdb) x アドレス
```

で、アドレスの内容をプリントすることができます。x のあとには、データ表示のフォーマットができて、例えば、x/のあとに、表示するデータの数、10 進 (d)、16 進 (x)、8 進 (o) とそのあとに、b(byte)、h(half)、w(word) と指定します。たとえば、

```
(gdb) x/10dw 0x10000
```

では、0x10000 番地から、32 ビットごと (w) に 10 進 (d) で、10 ワード表示するという意味になります。詳しくは、help x としてみてください。

他のコマンドについても、help コマンドで調べることができます。