基礎実験１　UNIX・アセンブラ実習 第５回

2007 年 4 月 23 日

実習内容

プログラムにバグがあるとき，レジスタやメモリの変化を観察したり，68000 エミュレータの機能であるブレークポイントを使って，バグを解決するための手がかりを得ることができます．今日の実習では、自分でアセンブラプログラムを入力し、そのプログラムのテスト実行とバグの発見，デバックを行います。

１．プログラムの入力

プログラムの入力には前回までの実習で使った emacs を使います。今回、実行するプログラムのファイル名は fact.s です。そこで、Terminal のウインドウ上で、「emacs fact.s &」 と入力しましょう。

```
$ emacs fact.s & ＜Enter キー＞
```

emacs のウインドウが開くのを待ちます。

それでは、実際に下記のプログラムファイル fact.s を emacs を使って作成しましょう．このプログラムは，バグの発見とデバッグを練習するために**故意に間違い**を入れてあります。

（注意）Pascal や C のような高級言語だと、たったの 1 行で済むような処理でも、アセンブラプログラムでは、何行にもなりプログラムも読みづらくなる。そのため、見やすい（読みやすくて分かりやすい）プログラムを心掛ける事が重要となる（アセンブラに限ったことではないが)。例えば、コメント文の活用、インデント（行頭の位置）の調節、シンボル名の付け方、Tab を使った単語の位置整列などを利用すると良い。

```
/* sample program fact.s 階乗  d1 := d0 ! (d0 >= 0)*/
.org   0x0000
.dc.l  0x5000
.dc.l  start

.org   0x0400
start:
       moveq  #1,%d1        /* d1 := 1; */
       cmp    #0,%d0        /* d0 が 0 なら最後へ*/
       beq    end_of_program
       move.l %d0,%d2       /* d2 はカウンター*/
loop:
       mulu   %d2,%d1       /* d1 := d1 * d2; */
       subq   #1,%d2        /* d2 := d2 - 1; */
       cmp    #0,%d2        /* d2 と 0 を比較 */
       bge    loop          /* d2 が 0 になるまでループ */
end_of_program:
       .dc.w  0x4848
       stop   #0            /* 終了 */
```

入力が終わったらファイルを保存しましょう。emacs 上で、

Ctrl-x Ctrl-s（上書き保存） または **Ctrl-x Ctrl-w**（別名で保存）

ファイルの中身を jless などのコマンドで確認しましょう。

```
$ less fact.s  ＜Enter キー＞
```

課題１．プログラム fact のフローチャートを考えて、解答せよ。

2. アセンブラ

前回の演習と同様に，m68k-as コマンドを使って，アセンブラソースプログラムファイル fact.s をアセンブルしましょう。kterm 上で次のコマンドを実行してみましょう。

```
$ m68k-as fact.s  ＜Enter キー＞
```

もし，文法的な間違いがある場合は，以下のようなエラーメッセージが出ます（あくまで例です）。その場合は，エラーメッセージを手がかりに emacs で fact.s ファイルを修正しましょう。

```
fact.s: Assembler messages:
fact.s:9: Error: parse error -- statement `cmp #0.%d0' ignored
fact.s:12: Error: Unknown operator -- statement `loop ' ignored
```

エラーメッセージがなくなったら、実行ファイルが出来ているか ls コマンドで確認しましょう。

```
% ls fact.*  ＜Enter キー＞
fact.LIS     fact.abs     fact.map     fact.s
```

（単に「ls ＜Enter キー＞」と実行すると関係ないファイルも表示されるため、「fact」で始まるファイルだけを表示させた）

3. エミュレータ

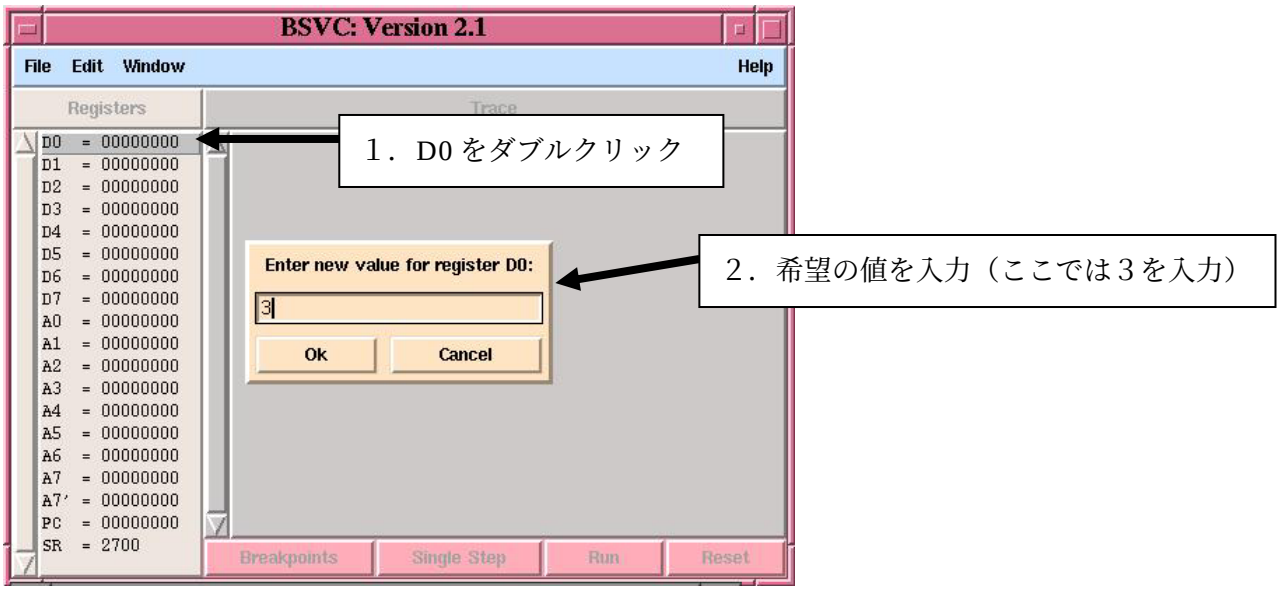
次に，68000 エミュレータ(m68k-emu)を実行しましょう。

```
% m68k-emu &  ＜Enter キー＞
```

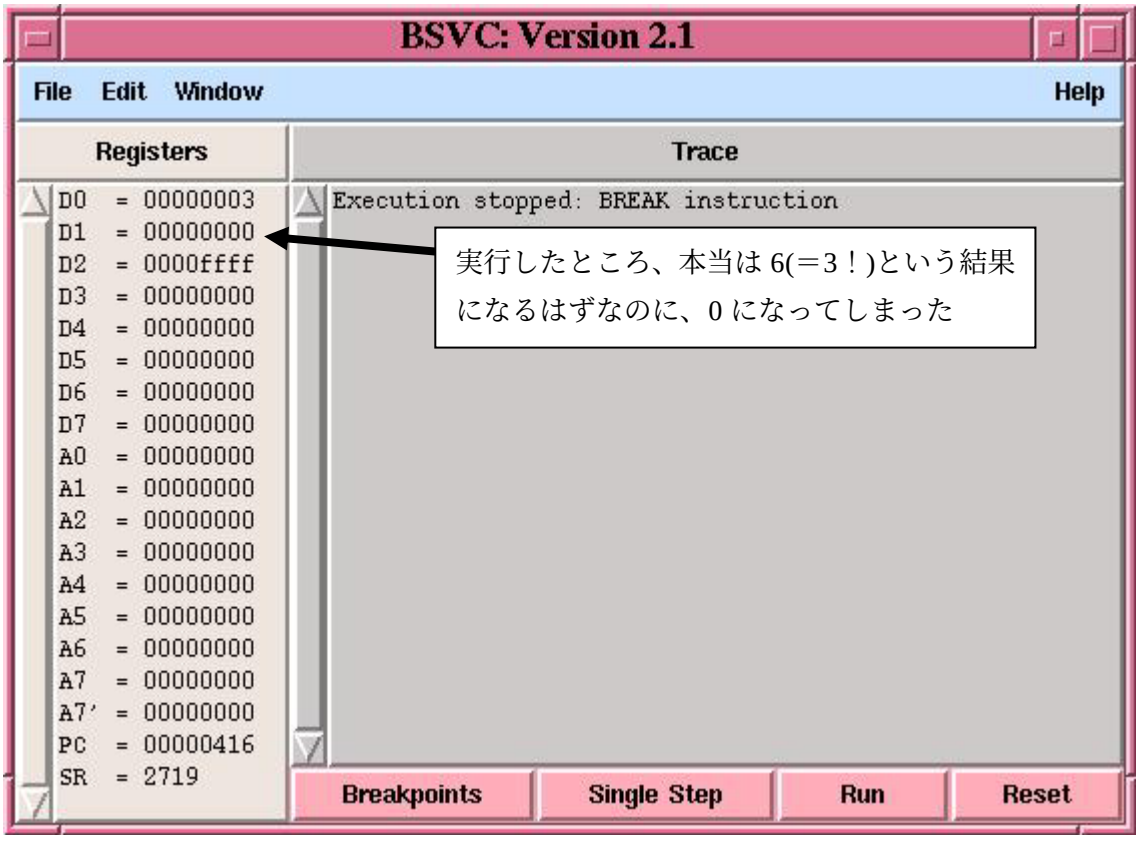
File メニューから、Load Program を実行し実行したいプログラムを選びます。ここでは、fact.abs

を選択してください。Window メニューの、Memory Viewer、 Program Listing を実行すると、アセンブラプログラムの実行の様子がモニターできます。

このプログラムは、レジスタ d0 の値の階乗を計算しレジスタ d1 に格納します。レジスタの値は画面の左にあるレジスタをダブルクリック（２度続けてマウスのボタンをたたく）し、値を入力することによって値が変更できます（下図)。3 の階乗を計算させて見ましょう。d0 に 3 を設定します。



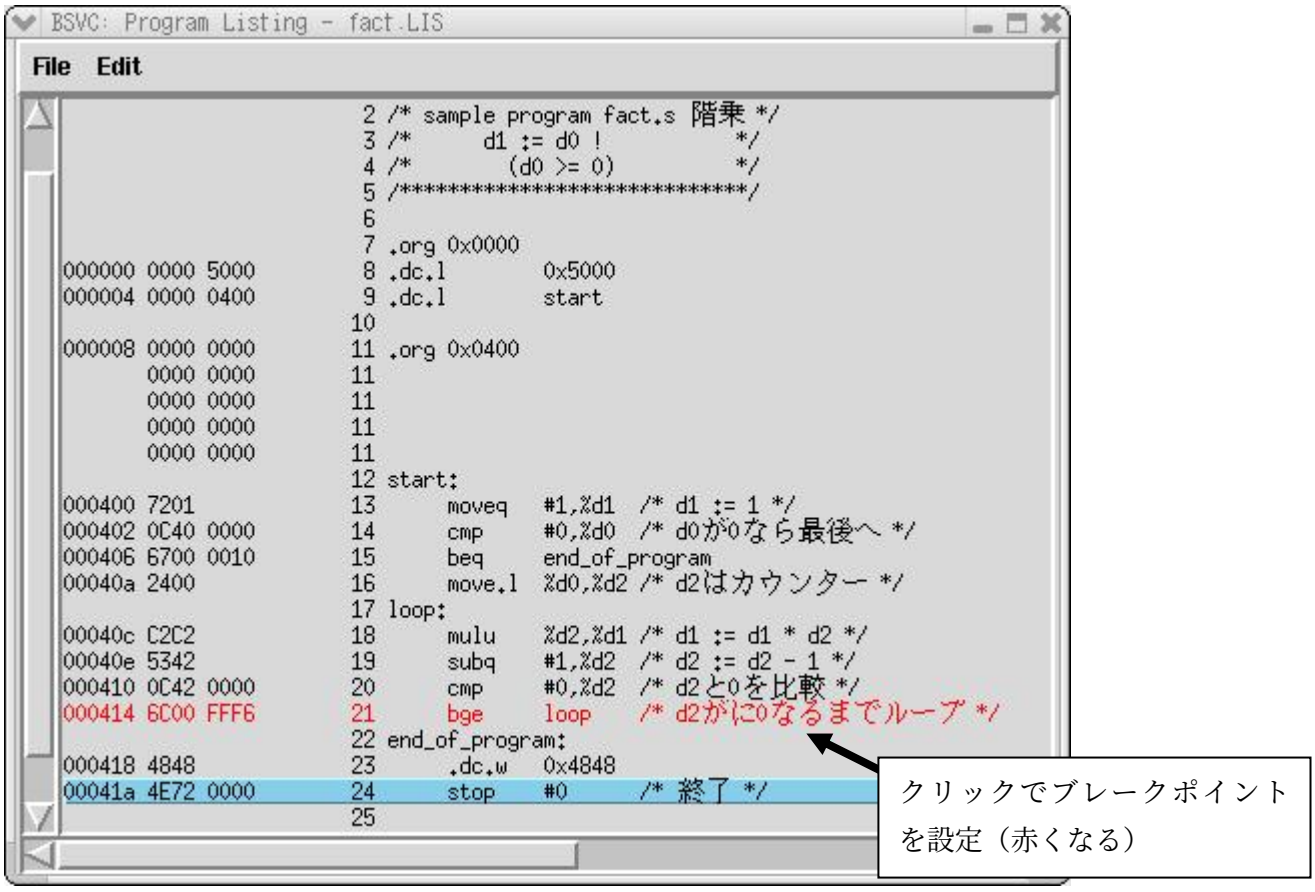
Run ボタンを押して実行してみましょう。計算結果が格納されるレジスタ D1 は０になってしまいました。プログラムが期待通りには動いていないことが分かりました. プログラム中にバグが残っています. データレジスタ D1 が０になってしまう原因を，これから探していきます.



4．ブレークポイント

プログラムを実行する際に、実行途中でメモリやレジスタの値を確認したい場合には、ブレークポイントを利用すると便利です。ブレークポイントを設定すると Run ボタンを押したときにブレークポイントを設定した箇所で、プログラムは一時的に停止します。

ブレークポイントを設定してみましょう。エミュレータのメインウィンドウにある Breakpoints ボタンを押して番地を設定することもできますが、Program Listing のウインドウでブレークポイントを設定したい命令をクリックする方が簡単です。ブレークポイントが設定されると、その命令箇所は赤い文字で表示されます。今回は、ループの分岐の際のレジスタを調べるために、ループの分岐命令(bge loop)をクリックしましょう。bge loop の行が赤くなりました。



Reset ボタンを押した後で Run ボタンを押して実行しましょう。ブレークポイントの位置で実行がとまりました。この時点でレジスタは

D0: 3　D1: 3　D2: 2

でした。続けて Run ボタンを 1 回ずつ押すと

D0: 3　D1: 6　D2: 1

D0: 3　D1: 6　D2: 0

D0: 3　D1: 0　D2: ffff

と変化しました。どうやら D2 が 0 のときにもループしてしまい D1 が 0 になってしまったようです。つまり、分岐の条件が間違っていたのです。

条件分岐の代表的なものをあげてみます。

bge	bgt	beq	ble	blt	bne
>=	>	=	<=	<	≠

bge loop を正しいものに置き換えて動かしてみましょう。

emacs でファイルを編集し、（終了していなければファイルを読み直しましょう）

```
$ emacs fact.s &  ＜Enter キー＞
```

再度、アセンブラを実行し、

```
$ m68k-as fact.s  ＜Enter キー＞
```

68000 エミュレータを実行しましょう。（終了していなければファイルを読み直しましょう）

```
$ m68k-emu & ＜Enter キー＞
```

d0 が 3 のときに d1 は 6 になりましたか？

4!,2!,1!,0!についても試してみましょう。Reset ボタンを押し、d0 レジスタに値を入れた後で、Run ボタンを押してください。4 では d1 は 18 と表示されたと思います。これは 16 進数で表示されているため 16*1+8=24=1*2*3*4 で正解です。

5. CCR（SR）と条件付分岐命令

今回のプログラムのサブルーチン中には，下のような条件付分岐命令が含まれている（１）。しかしながら，この条件分岐命令 cmp は必要ではなく，（２）のように省くことができる．なぜか？

```
(1)
        subq   #1,%d2
        cmp.w  #0,%d2
        bge    loop
```

```
(2)
        subq   #1,%d2
        bge    loop
```

この理由を理解するためには、まず cmp 命令の意味を理解する必要がある。cmp 命令は実際にはデスティネーションオペランドからソースオペランドを減算している（命令表 p.310 を必ず見ること）。そして、その結果は CCR（コンディションコードレジスタ）に反映される。この CCR とは、SR（ステータスレジスタ）の下位８ビット（実際に使うのは 5 ビット）のことであり、エミュレータメインウィンドウ左のレジスタ一覧にも表示されている。この CCR の詳しい説明は以下の URL を参照すること

http://www.db.is.kyushu-u.ac.jp/rinkou/as/advanced/ccr.html

つまり、大小比較のため減算をし、その結果が負ならば CCR の N のビットを１にし、結果が０ならば CCR の Z のビットを１に設定している。もちろん、結果が正の場合は、CCR の N と Z のビットは０のままとなる。こうすることで、CCR を見れば、直前に行われた比較（減算）結果がどうであったかが分かる仕組みになっているのである。**cmp 命令が sub 命令と違う点は、減算した結果がデスティネーションオペランドには格納されないという点**だけであり、sub 命令も減算の結果、cmp 命令と同様に CCR に影響を及ぼす。

次に、条件付分岐命令 bcc（cc には不等号条件が設定される）（命令表 p.372 を必ず見ること）の意味について説明する。この命令は、CCR に応じて必要なロケーションへ制御を移行させるものである。そのため、本例の場合、レジスタ d2 から 1 を減算した結果が 0 でなければ、CCR の Z、N のビットが 0 となり、bge 命令はその CCR を参照した結果、loop に制御を移行させるのである。そのため、（2）のプログラムのように，sub 命令と bge 命令の間の cmp 命令を省くことができるのである。
この一連の流れを確認するために、分岐命令の箇所（1）を（2）のように変更し，条件分岐命令付近での CCR（実際には SR の下位 5 ビット）の変化を確認しましょう。
*（注意）subq 命令と sub 命令の違い、moveq 命令と move 命令の違いについては、命令表を自分で調べてみること。**自分で積極的に試したり、調べることも実習の一部です。***

＜印刷方法＞　http://brain.is.kyushu-u.ac.jp/~matsuki/enshu/2007/print.htm を参考にすること

課題２．fact.s を参考にして、分岐命令（繰り返し）を使ったプログラムを作成せよ．そして，その計算結果を報告せよ．

（1）$_{10}P_5$

（2）$\sum_{k=1}^{5} k^k$

今日の実習はここまでです。

参考 Web ページ: http://www.db.is.kyushu-u.ac.jp/kaneko/as/index.html