システムプログラミング序論 後半 アセンブリ言語編 演習 0

Linux/x86 環境のマシン

この講義の後半の回では演習に Linux/x86 のマシンを使う．情報科学類コンピューティング環境で Linux/x86 のマシンを使う選択肢は複数ある．第一の選択肢は，サーバ viola12.coins.tsukuba.ac.jp に SSH でリモートログインして Linux を使うことである．第二の選択肢は，計算機室の端末で macOS ではなく Linux を起動して Linux を使うことである．SSH やリモートログインの詳細については，情報科学類が PDF ファイルの形で公開している「教育用計算機システム仕様の手引き」を読んでほしい．

計算機室の外での作業

情報科学類コンピューティング環境のファイルシステムに SSH の公開鍵を置いておくと，学内のどこからでも，自宅からでも，国外からでも，サーバや端末にリモートログインすることができる．すでにそうしている人も多いと思うが，まだしておらず，自宅などの計算機室の外でも演習を行いたいと思っている人は，リモートログインのための作業をしておくと良いだろう．

自前の Linux 環境を構築することもできる．VMware Workstation Player，VirtualBox，Parallels Desktop，KVM，QEMU，Docker などのソフトウェアを用いると，Windows や macOS などの OS 上に仮想マシンやコンテナを作り，その上で Linux を使用することができる．

演習で用いる gcc の 32 ビット開発環境を構築する方法については，例えば，「gcc 32 ビット 開発環境 インストール」などのキーワードで Web を検索すると，多くの情報を得られる．Ubuntu を使う場合にはキーワードに apt と install，CentOS を使う場合にはキーワードに yum と install を加えて検索するとよい．

情報科学類の学生は多くのソフトウェアを無償で利用することができる．詳しい情報は以下の Web サイトに掲載されている．

情報科学類コンピューティング環境
https://www.coins.tsukuba.ac.jp/ce/

文字コード

Linux と macOS では文字コードが異なる場合があり，SSH でリモートログインした後のエラーメッセージ等が文字化けするかもしれない．これに対する一つの対策は，英語でエラーメッセージを出力するようにして，その英語のエラーメッセージを理解することである．その対策を実行するには，たとえば．リモートログインした後で以下のコマンドを実行すればよい．

```
$ export LANG=C
```

これは，文字コードを無効（常に英語）にするという意味を持つ．もしあくまで日本語で表示させたい場合には，自分で Web を検索したりしてその方法を調べてほしい．

ファイルの終端コード

アセンブリ言語のソースプログラムを書いて，gcc コマンドで，機械語（バイナリ）コードが書かれた実行型ファイルに変換しようとすると，以下のメッセージが出ることがある．

```
Warning: end of file not at end of a line; newline inserted
```

これは，「ファイルの最後に改行がないままファイルが終わってしまったので，改行を入れておいたよ」という警告である．ファイルの最後には必ず改行を 1 つ入れておかなければならない．ただ，コマンドはこの警告を出した後，改行を自動的に入れてくれているので，変換自体は正常に進む．

16 進電卓

課題を解く際，自分の書いたアセンブリ言語のプログラムが出す答が正しいのかどうか，検算をしたくなることがある．検算には例えば「16 進電卓」が使える．Windows や macOS に標準で入っている電卓には 16 進数で計算する機能がある．UNIX にはテキストベース電卓のコマンドとして bc があり，これは「入力／出力の基数を設定する」という機能を持っている．以下のように bc を実行すると，入力を 16 進数で与え，結果も 16 進数で表示する」という電卓を使うことができる．

```
$ bc
```

(何かメッセージが出てくるが，気にしない)

```
obase=16
ibase=16
```

このように ibase と obase を設定した後で，例えば

```
1F+35
```

と入力すると，結果が 54 と表示される．bc コマンドを終了するには Ctrl-D をタイプする．ibase=は「入力の基数を変更する」という意味である．これ以降の入力は指定された値を基数とする「進数」で表される．obase=は同様に出力の基数を変更する．obase を指定する前に ibase を指定するとおかしなことになるので注意する（なぜそうなるのかは自分で考えてほしい）．対話的な実行が可能なプログラミング言語処理系を使うのもよい．例えば，irb コマンドによって，Ruby のプログラムを対話的に実行できる．

```
$ irb
irb(main):001:0> 0xab+0x12
=> 189
irb(main):002:0> (0xab+0x12).to_s(16)
=> "bd"
irb(main):003:0>
```

また，python や python3 コマンドによって，Python のプログラムを対話的に実行できる．

```
$ python3
Python 3.6.5 |Anaconda custom (64-bit)| (default, Apr 29 2018, 16:14:56)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 0xab+0x12
189
>>> hex(0xab+0x12)
'0xbd'
>>>
```

課題 0（どれも提出不要）

課題 0-1

どこでもいいので Linux 環境にログインし，自分がこの科目の前半の回で書いた C プログラムのどれかを，gcc コマンドによって x86（32 bit）のアセンブリ言語のプログラムに変換せよ．gcc には-S -m32 というオプションを与える．拡張子が.c であるファイルから，拡張子が.S であるファイルが作られるはずである．変換後のプログラムをざっと見て，プログラムの各部分が何を表す情報であるかを推測せよ．環境によっては gcc がインストールされていなかったり，-m32 というオプションのためのソフトウェアがインストールされていなかったりするかもしれない．その場合には，必要なソフトウェアをインストールしたり，すでにそれらがインストールされた環境に移ったりしてほしい．今後の演習では，-m32 というオプション付きで gcc を使うので，自分が使う予定の環境でその処理を実行できることは極めて重要である．

課題 0-2

好きな C プログラムを gcc で-g -m32 というオプションをつけてコンパイルして実行可能ファイルを生成せよ．その実行可能ファイルを，デバッガ gdb 上で実行せよ．デバッガはプログラムをデバッグするためのツールである．デバッグしたいプログラムを指定の場所で止めたり，少しずつ実行したり，そのプログラムの変数やメモリに入っている値を読んだりすることができる．

実行可能ファイルの名前を a.out とすると，以下のコマンドによって a.out を gdb 上で実行できる．

```
$ gdb ./a.out
```

gdb の実行例をこの資料末尾の付録に示す．実行例で実行している gdb のコマンドすべてを自分でも gdb 内で実行し，それらが正常に実行されている（らしき）ことを確認せよ．実行例で gdb のコマンドの引数に与えている関数名などの情報は適宜自分のプログラムのそれに置き換えよ．また，実行例で用いているコマンドのヘルプメッセージを読み，実行例の各コマンドの意味を理解するとともに，できるだけ多くの gdb のコマンド名と機能を記憶せよ．

実行例を読んで理解すれば自分で実行しなくてもいいと思ってはいけない．自分で手を動かしてシステムを実際に操作する（いわば，「写経」する）ことが，記憶を定着させシステムの使い方を習得するためには重要である．

課題 0-3

すでに OS にインストールされているコマンドやアプリケーションのどれか（/bin/ls, /usr/bin/gnome-terminal, /usr/bin/vim, /usr/bin/ssh など）を選び，それらのアセンブリコードを出力して眺めてみよ．実行可能ファイルを foo とすると，objdump –d foo というコマンドによってアセンブリコードを出力することができる．

課題 0-4

最適化のオプション（-O）やデバッグ情報のオプション（-g）を様々に変えて，C プログラムからアセンブリ言語プログラムを生成せよ．上記のオプションの詳細についてはマニュアルを参照せよ．生成された複数のアセンブリ言語プログラムを比較し，それらの間で何が違っているかを見つけ，その違いがもたらす効果を推測せよ．

課題 0-5

Web の検索などにより，x86 アセンブリ言語の学習や x86 の仕様の参照に役立ちそうな Web ページを見つけよ．それらの Web ページをブックマークに追加するなどのことを行い，それらの Web ページにすぐに行ける状態にせよ．例えば，以下の Web ページを見てみよ．

- 演習で用いるアセンブラ GNU as (gas) のマニュアル
  https://sourceware.org/binutils/docs-2.27/as/index.html
- アセンブラ入門
  http://programmer.main.jp/assembler/assembler00.html
- Wikibooks: X86 アセンブラ
  https://ja.wikibooks.org/wiki/X86%E3%82%A2%E3%82%BB%E3%83%B3%E3%83%96%E3%83%A9

課題 0-6（発展課題）

Intel 社の Web サイト

Intel® 64 and IA-32 Architectures Software Developer Manuals
https://software.intel.com/en-us/articles/intel-sdm

に行き，”Intel® 64 and IA-32 architectures software developer's manual combined volumes 2A, 2B, 2C, and 2D: Instruction set reference, A-Z”などの x86 に関するマニュアルをダウンロードせよ（ファイルサイズが大きくページ数が多いことに注意）．目次や中身をざっと眺めたり拾い読みしたりして，どんな情報が書かれている（または書かれていそう）かを理解せよ．

日本語に翻訳されたマニュアルも Web 上に存在するが，バージョンが非常に古い．

課題 0-7（発展課題）

x86 アセンブリ言語の Intel 記法と AT&T 記法について調べよ．gdb や objdump により，両方の記法でア

センブリコードを生成し，それらを比較せよ．gdb では set disassembly-flavor intel というコマンドで Intel 記法に切り替えることができ，set disassembly-flavor att というコマンドで AT&T 記法に切り替えることができる．objdump では-d -M intel というオプションで出力を Intel 記法にすることができ，-d –M att というオプションで出力を AT&T 記法にすることができる．

課題 0-8（発展課題）

整数の定数 2 つを加算して和を表示する単純な C プログラムを記述せよ．その C プログラムのファイルを foo.c とする．gcc –m32 –S foo.c というコマンドを実行することによって．アセンブリコードのファイル foo.s を生成せよ．foo.s の命令列のうち，C プログラムでの定数に対応していると思われる中の部分をエディタなどで編集して，別の定数に書き換えよ．書き換え後のプログラムをファイルに保存せよ．保存したファイル名を foo2.s とする．gcc –m32 foo2.s というコマンドを実行することによって実行可能ファイルを生成し，それを実行せよ．書き換え後の定数の値で加算が行われることを確認せよ．

A. 付録： gdb の実行例

```
$ hostname -A
viola12.coins.tsukuba.ac.jp viola12.coins.tsukuba.ac.jp
$ date
Sat Dec  7 12:00:00 JST 2019
$ gcc -v
組み込み spec を使用しています。
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/lto-wrapper
ターゲット: x86_64-redhat-linux
configure 設定: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --
with-bugurl=http://bugzilla.redhat.com/bugzilla  --enable-bootstrap  --enable-shared  --enable-
threads=posix  --enable-checking=release  --with-system-zlib  --enable-__cxa_atexit  --disable-
libunwind-exceptions  --enable-gnu-unique-object  --enable-linker-build-id  --with-linker-hash-
style=gnu  --enable-languages=c,c++,objc,obj-c++,java,fortran,ada,go,lto  --enable-plugin  --
enable-initfini-array       --disable-libgcj       --with-isl=/builddir/build/BUILD/gcc-4.8.5-
20150702/obj-x86_64-redhat-linux/isl-install     --with-cloog=/builddir/build/BUILD/gcc-4.8.5-
20150702/obj-x86_64-redhat-linux/cloog-install      --enable-gnu-indirect-function      --with-
tune=generic --with-arch_32=x86-64 --build=x86_64-redhat-linux
スレッドモデル: posix
gcc バージョン 4.8.5 20150623 (Red Hat 4.8.5-39) (GCC)
$ gdb -v
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-115.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
$ cat plus.c
#include <stdio.h>

int plus(int x, int y)
{
  int z;
  z = x + y;
  return z;
}

int main(void)
{
  int a, b, c;
  a = 3;
  b = 6;
  printf("%d + %d = ", a, b);
  c = plus(a, b);
  printf("%d¥n", c);
  return 0;
}
$ gcc -g -m32 plus.c
$ gdb ./a.out
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-115.el7
Copyright (C) 2013 Free Software Foundation, Inc.
```

```
...
 (gdb) disas plus
Dump of assembler code for function plus:
   0x0804840d <+0>:     push   %ebp
   0x0804840e <+1>:     mov    %esp,%ebp
   0x08048410 <+3>:     sub    $0x10,%esp
   0x08048413 <+6>:     mov    0xc(%ebp),%eax
   0x08048416 <+9>:     mov    0x8(%ebp),%edx
   0x08048419 <+12>:    add    %edx,%eax
   0x0804841b <+14>:    mov    %eax,-0x4(%ebp)
   0x0804841e <+17>:    mov    -0x4(%ebp),%eax
   0x08048421 <+20>:    leave
   0x08048422 <+21>:    ret
End of assembler dump.
(gdb) disas main
Dump of assembler code for function main:
   0x08048423 <+0>:     push   %ebp
   0x08048424 <+1>:     mov    %esp,%ebp
   0x08048426 <+3>:     and    $0xfffffff0,%esp
   0x08048429 <+6>:     sub    $0x20,%esp
   0x0804842c <+9>:     movl   $0x3,0x1c(%esp)
   0x08048434 <+17>:    movl   $0x6,0x18(%esp)
   0x0804843c <+25>:    mov    0x18(%esp),%eax
   0x08048440 <+29>:    mov    %eax,0x8(%esp)
   0x08048444 <+33>:    mov    0x1c(%esp),%eax
   0x08048448 <+37>:    mov    %eax,0x4(%esp)
   0x0804844c <+41>:    movl   $0x8048524,(%esp)
   0x08048453 <+48>:    call   0x80482e0 <printf@plt>
   0x08048458 <+53>:    mov    0x18(%esp),%eax
   0x0804845c <+57>:    mov    %eax,0x4(%esp)
   0x08048460 <+61>:    mov    0x1c(%esp),%eax
   0x08048464 <+65>:    mov    %eax,(%esp)
   0x08048467 <+68>:    call   0x804840d <plus>
   0x0804846c <+73>:    mov    %eax,0x14(%esp)
   0x08048470 <+77>:    mov    0x14(%esp),%eax
   0x08048474 <+81>:    mov    %eax,0x4(%esp)
   0x08048478 <+85>:    movl   $0x804852f,(%esp)
   0x0804847f <+92>:    call   0x80482e0 <printf@plt>
   0x08048484 <+97>:    mov    $0x0,%eax
   0x08048489 <+102>:   leave
   0x0804848a <+103>:   ret
End of assembler dump.
(gdb) list plus
1       #include <stdio.h>
2
3       int plus(int x, int y)
4       {
5         int z;
6         z = x + y;
7         return z;
8       }
9
10      int main(void)
(gdb) list main
6         z = x + y;
7         return z;
8       }
9
10      int main(void)
11      {
12        int a, b, c;
13        a = 3;
14        b = 6;
15        printf("%d + %d = ", a, b);
(gdb)
16        c = plus(a, b);
17        printf("%d¥n", c);
18        return 0;
19      }
(gdb) break plus
Breakpoint 1 at 0x8048413: file plus.c, line 6.
(gdb) break main
Breakpoint 2 at 0x804842c: file plus.c, line 13.
(gdb) break plus.c:15
Breakpoint 3 at 0x804843c: file plus.c, line 15.
(gdb) run
Starting program: ...

Breakpoint 2, main () at plus.c:13
13        a = 3;
Missing separate debuginfos, use: debuginfo-install glibc-2.17-292.el7.i686
(gdb) disas main
```

```
Dump of assembler code for function main:
   0x08048423 <+0>:     push   %ebp
   0x08048424 <+1>:     mov    %esp,%ebp
   0x08048426 <+3>:     and    $0xfffffff0,%esp
   0x08048429 <+6>:     sub    $0x20,%esp
=> 0x0804842c <+9>:     movl   $0x3,0x1c(%esp)
   0x08048434 <+17>:    movl   $0x6,0x18(%esp)
   ...
End of assembler dump.
(gdb) c
Continuing.

Breakpoint 3, main () at plus.c:15
15      printf("%d + %d = ", a, b);
(gdb) backtrace
#0  plus (x=3, y=6) at plus.c:6
#1  0x0804846c in main () at plus.c:16
(gdb) frame 0
#0  plus (x=3, y=6) at plus.c:6
6       z = x + y;
(gdb) info frame 0
Stack frame at 0xffffd280:
 eip = 0x8048413 in plus (plus.c:6); saved eip 0x804846c
 called by frame at 0xffffd2b0
 source language c.
 Arglist at 0xffffd278, args: x=3, y=6
 Locals at 0xffffd278, Previous frame's sp is 0xffffd280
 Saved registers:
  ebp at 0xffffd278, eip at 0xffffd27c
(gdb) frame 1
#1  0x0804846c in main () at plus.c:16
16      c = plus(a, b);
(gdb) info frame 1
Stack frame at 0xffffd2b0:
 eip = 0x804846c in main (plus.c:16); saved eip 0xf7e102a3
 caller of frame at 0xffffd280
 source language c.
 Arglist at 0xffffd2a8, args:
 Locals at 0xffffd2a8, Previous frame's sp is 0xffffd2b0
 Saved registers:
  ebp at 0xffffd2a8, eip at 0xffffd2ac
(gdb) print a
$1 = 3
(gdb) print b
$2 = 6
(gdb) next
16      c = plus(a, b);
(gdb) stepi
0x0804845c      16      c = plus(a, b);
(gdb)
0x08048460      16      c = plus(a, b);
(gdb)
0x08048464      16      c = plus(a, b);
(gdb)
0x08048467      16      c = plus(a, b);
(gdb)
plus (x=3, y=6) at plus.c:4
4       {
(gdb)
0x0804840e      4       {
(gdb)
0x08048410      4       {
(gdb)

Breakpoint 1, plus (x=3, y=6) at plus.c:6
6       z = x + y;
(gdb) disas plus
Dump of assembler code for function plus:
   0x0804840d <+0>:     push   %ebp
   0x0804840e <+1>:     mov    %esp,%ebp
   0x08048410 <+3>:     sub    $0x10,%esp
=> 0x08048413 <+6>:     mov    0xc(%ebp),%eax
   0x08048416 <+9>:     mov    0x8(%ebp),%edx
   ...
End of assembler dump.
(gdb) stepi
0x08048416      6       z = x + y;
(gdb) print x
$3 = 3
(gdb) print y
$4 = 6
(gdb) print z
```

```
$5 = -134227712
(gdb) info reg
eax            0x6     6
ecx            0x0     0
edx            0xf7fbe898      -134485864
ebx            0xf7fbd000      -134492160
esp            0xffffd268      0xffffd268
ebp            0xffffd278      0xffffd278
esi            0x0     0
edi            0x0     0
eip            0x8048416       0x8048416 <plus+9>
eflags         0x282   [ SF IF ]
...
 (gdb) info reg eax
eax            0x6     6
(gdb) info reg eip
eip            0x8048416       0x8048416 <plus+9>
(gdb) info reg pc
pc             0x8048416       0x8048416 <plus+9>
(gdb) stepi
0x08048419     6        z = x + y;
(gdb)
0x0804841b     6        z = x + y;
(gdb)
7       return z;
(gdb) print z
$6 = 9
(gdb) info reg
eax            0x9     9
ecx            0x0     0
edx            0x3     3
ebx            0xf7fbd000      -134492160
esp            0xffffd268      0xffffd268
ebp            0xffffd278      0xffffd278
esi            0x0     0
edi            0x0     0
eip            0x804841e       0x804841e <plus+17>
eflags         0x206   [ PF IF ]
...
 (gdb) stepi
8      }
(gdb)
0x08048422     8      }
(gdb)
0x0804846c in main () at plus.c:16
16      c = plus(a, b);
(gdb) info reg esp
esp            0xffffd280      0xffffd280
(gdb) info reg sp
sp             0xffffd280      0xffffd280
(gdb) x/8x $esp
0xffffd280:    0x00000003      0x00000006      0x00000006      0xf7e289bd
0xffffd290:    0xf7fbd3c4      0x00008000      0x00000006      0x00000003
(gdb) x/20x $esp+32
0xffffd2a0:    0x08048490      0x00000000      0x00000000      0xf7e102a3
0xffffd2b0:    0x00000001      0xffffd344      0xffffd34c      0xf7fd86b0
0xffffd2c0:    0x00000001      0x00000001      0x00000000      0x0804a010
0xffffd2d0:    0x0804821c      0xf7fbd000      0x00000000      0x00000000
0xffffd2e0:    0x00000000      0x73565bfe      0x4ef7ffee      0x00000000
(gdb) next
17      printf("%d¥n", c);
(gdb)
3 + 6 = 9
18      return 0;
(gdb) cont
Continuing.
[Inferior 1 (process 11593) exited normally]
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
...
 (gdb) help running
Running the program.

List of commands:

advance -- Continue the program up to the given location (same form as args for break command)
...
 (gdb) help stepi
```

```
Step one instruction exactly.
Usage: stepi [N]
Argument N means step N times (or till program stops for another reason).
(gdb) quit
$
```