Assignment 1: Basic ARM32 assembly and argument handling

Due: Monday 09/14/20 (before class)

In this assignment, you will write a basic ARM32 assembly routine using gnu assembly. To log into first time, do the following:

1. Log into the frontend of my research machines. Assuming start machine Unix (eg. BSD, OS X or Linux): `ssh -X UNAME@etl.sice.indiana.edu`, replacing UNAME with your IU login. (Windows users can use putty with X forwarding enabled)
2. If you want, change to IU password using *passwd* and/or setup ssh-keys
3. From there, log into 32-bit arm machine vi `ssh a15`
4. If you want, change to IU password using *passwd* and/or setup ssh-keys
5. You can set ssh to allow you do directly ssh or scp to/from `a15` if you set up ssh-keys, see end of handout for how.

On `a15`, change into a directory you create for this assignment, and issue the following command to get the file you need:

```
cp ~whaley/classes/ARMF20/p1_istr/Makefile .
```

View and understand the provided Makefile and `tester.c` so that you can write your own in the future, or adapt the Makefile to work on your home system.

In this assignment you will write an assembly program in the required filename of `istr.S`. The file will implement a function with the prototype:

```
int istr(int i1, int i2, int i3, int *i4, char *alp);
```

This function will do three things:

1. Return `i1 + i2`,
2. Set *i4 = 5 * (*i4) + i3;,
3. and write the lower-case alphabet to the string `alp`
   - See the `tester.c` for how the string is declared and the function call is made
   - Remember that a C string must end with the string terminator; for more on strings, see slide 20 of the C lectures:

     `http://homes.sice.indiana.edu/rcwhaley/teach/iseARM_F20/stuff/02varMem_ho.pdf`

I suggest you get this working in three distinct steps:

1. First, write a function taking the above arguments, but the only thing it does is return i1 + i2. You can test this function by issuing '`make xitest0`' to build things, and '`./xitest0`' to run the executable once the build works.
2. Now that the basics are working, try to use the address passed in as the `i4` parameter to change the caller's variable. You can test that both integer operations pass sanity checks by issuing: '`make xitest`' and '`./xitest`'.
3. Once the integer code is mastered, implement the alphabet string portion of the assignment. Sanity check the completed function with '`make xtest`' and '`./xtest`'.
   - If you use a loop to do this, you will lose 4 points for every extra instruction beyond 4 within the loop, and get +5 for every instruction less

  - If you don't use a loop, I will make fun of you, and then count off -4 for every instruction over 9 you take
  - → Despite above threats, write it in simplist way first, then improve it once you know you have the idea working!

In addition, make sure that:

- You scope `tester.c`, and see how CPP macros are used to form all the subtesters from one source file
- Scope how the Makefile compiles the same file with differing flags in order to build the different targets
- Try to generalize this knowledge for your own use

REQUIRED CODING STYLE

1. No use of `push` or `pop`
2. Use self-documenting cpp-defines in place of register names whenever practical
   - Exceptions for `SP` (recommended) and `r0` (**only** when used for int return)
   - → May make sense only for string code in this asg
3. Lines must be $\leq$ 80 columns including all white space and comments
4. Do not use tabs
5. Use indentation (miminum of 3 columns) to show code structure
6. Must comment code for maintainability (`//` and `/* */` OK)
   - Until told otherwise, provide a comment for each code line

ASSEMBLY TIPS

- Write first on paper, then type, then test.
- For every error, examine what you messed up, so you can write correct assembly on forthcoming quiz.
  - – If more than slight changes are required, start again with fresh sheet of paper, until you understand it wholly.
- When compiling assembly by hand, give it the extension **.S** (capital `S` means assembly with cpp), and compile with `gcc -marm -save-temps -g` (can leave off `-save-temps -g` if you aren't running debugger).
  - → Assuming tester is in `tester.c`, and assembly in `example.S`, we could build executable with:
    `gcc -marm -g -save-temps -o xtest tester.c example.S`
    The meaning of these arguments to `gcc` are:

`-marm`	: do pure arm code, no thumb
`-g`	: I want debugging output so I can use ddd or gdb to debug the executable
`-save-temps`	: tells compiler to keep all temporary files around, often necessary to get good debugging for assembly
`-o`	name the executable the name I give you after this flag
`tester.c`	filename, gcc assumes by .c extension it is a C file ⇒ compile
`example.S`	filename, gcc assumes by .S to be an assembly file with cpp commands ⇒ run `cpp` then assemble

AVOIDING TWO-HOP LOGIN

Since `a15` is a research machine, it has to be kept behind the gateway machine `etl.sice.indiana.edu` for security reasons. Having to always manually log into etl can make using a15 a pain, and so if you have set up ssh-keys to avoid typing in your passwords, you can have ssh handle the hop for you:

1. Add the following lines to your `~/.ssh/config`, adapting

   ```
   host etl
      User <your etl login>
      HostName etl.sice.indiana.edu
   ```

   If you have multiple ssh keys, you'll need to add a 3rd line right after these of form

   ```
      IdentityFile </path/to/the/id/file>
   ```

   Here is my entry for etl on my home machine:

   ```
   host etl
      User rcwhaley
      HostName etl.sice.indiana.edu
      IdentityFile /home/whaley/.ssh/id_rsa_iu
   ```

   This allows you to avoid typing the whole name, eg. `ssh etl` should now work.

2. You now need to add the entry for a15, and there are two methods depending on what version of ssh server you have:

   (a) The newer ssh servers would add (adapt to your info):

   ```
   host a15
      User whaley
      ProxyJump etl
   ```

   (b) I've got an old Linux install, so I needed to use (adapt to your info):

   ```
   host a15
      User whaley
      HostName 192.168.0.7
      ProxyCommand ssh -W %h:%p rcwhaley@etl
   ```