CS 241 – Week 1 Tutorial

Setup, Binary and Assembly Basics

Fall 2018

1 Setup

To access the student environment via the command line, you must ssh into your user account. Linux/Mac users can use Terminal and execute the command `ssh userid@linux.student.cs.uwaterloo.ca`. Windows users should use an ssh client called PuTTY, or Cygwin, or use a virtual machine.

1.1 Password-less ssh

Entering your password every time is tedious, but there are simple steps you can take to use password-less ssh. Instructions for those on Windows machines using PuTTY can be found at https://www.getfilecloud.com/blog/ssh-without-password-using-putty/. For Linux/Mac users, you can enter the two following commands:

```
ssh-keygen -t rsa
```

Keep hitting Enter until this completes. This will generate the two keys and put them in `~/.ssh/` with the names "`id_rsa`" for your private key, and "`id_rsa.pub`" for your public key.

```
scp ~/.ssh/id_rsa.pub userid@linux.student.cs.uwaterloo.ca:~/.ssh/authorized_keys
```

This will copy your public key to a new file called `authorized_keys`. Subsequent public keys can be appended to this file. This means that if you wanted to add another public key for another computer on this server, you would copy the contents of the second `id_rsa.pub` file into a new line on the existing `authorized_keys` file.

You now can now access the student environment from your machine without a password.

1.2 .profile

- When you log into the CS environment there are a number of files that get executed. One of these files is `~/.profile`.
- For convenience, edit .profile to execute the command `source /u/cs241/setup`. This will save you from having to source CS241 tools every time you ssh.
- Aliases, for commands you use often, should also be added to `.profile`. For example:

  alias xxd4='xxd -bits -cols 4'
- You can also write your own functions and make them available in .profile to simplify repeated tasks.

- Bonus tip for Mac users: You can similarly edit (or create) `~/.bash_profile` on your machine if you use bash and add the following alias to simply logging into to the student environment.
  `alias lse='ssh userid@linux.student.cs.uwaterloo.ca'`

1.3 Further reading

When it comes to editing files/programs, some of you prefer to use the student environment using Vim or Emacs (or Nano), while others prefer to edit on their local machines using a graphical editor such as Atom or Sublime Text. Note that CS241 tools are only available on the student environment.

For those of you who wish to edit files on their local machine, you may want to mount the student environment as a network drive on your computer, to avoid having to copy files to/from the student environment for testing. There are a variety of options, not limited to SSHFS (`https://www.digitalocean.com/community/tutorials/how-to-use-sshfs-to-mount-remote-file-systems-over-ssh`), Mountain Duck (paid), and Cyberduck (free). You are not required to use any of these, but it has been helpful and time-saving for students in the past.

2 Binary

2.1 The basics

To begin, we will review the most basic concept of data representation in computers, binary.

- Recall that binary data has no meaning on its own: we need to be told (or come up with) an interpretation for it. Give five different possible ways we could interpret 1101.
- Convert the 8-bit binary number 01101001 into decimal. (because of the leading 0, it could be unsigned or signed)
- What is the decimal range of unsigned n-bit binary numbers?

We can convert positive numbers into binary by repeated division by 2. For example, to convert 23 into binary:

- $23/2 = 11$ remainder 1
- $11/2 = 5$ remainder 1
- $5/2 = 2$ remainder 1
- $2/2 = 1$ remainder 0
- $1/2 = 0$ remainder 1

Then reading digits from bottom to top, we get $23_{10} = 10111_2$. To verify this, we can work backwards: $10111_2 = 2^4 + 2^2 + 2^1 + 2^0 = 16 + 4 + 2 + 1 = 23$. Leading digits are just 0, so in 8-bit binary we would write 00010111.

We can also convert positive numbers into binary by repeatedly subtracting the largest power of two.

- Convert the following numbers into unsigned 8-bit binary:
  1. 35
  2. 216
- Encode 35 and 1 in binary and compute the value $35 + 1$ in unsigned 8-bit binary.

2.2 Two's Complement

If we also want to be able to represent negative numbers as well as positive ones, we can use an encoding called "two's complement". The simplest way to use two's complement is to encode the number as if it were unsigned (so if you want to encode -123, encode 123), then apply the two's complement algorithm presented last week: flip all bits and add 1.

- Why does this work? Call x^* the result of flipping all the bits of x (so for example, if $x = 1011$ then $x^* = 0100$). Then $x + x^* = 1111 = 2^b - 1$. So since we picked $-x = x^* + 1$, we get $x + (-x) = x + x^* + 1 = 2^b - 1 + 1 = 2^b \equiv 0 \mod 2^b$.

The second way of encoding $-x$ is to instead encode $2^b - x$.

- Why does this work? $x + (-x) = x + 2^b - x = 2^b \equiv 0 \mod 2^b$.

Try the following in 8-bit binary:

1. Encode -12 using the two's complement algorithm.
2. Encode -123 using the second way above.
3. Convince yourself that, for example, $- - 15 = 15$ by either two's complement technique listed above. What is $- - 128$?
4. Encode -12 and 2 using the two's complement algorithm and add the two binary number.

Side Question: Why do we use two's complement instead of signed magnitude?

3 Assembly basics

3.1 Assembly instructions

Assembly languages are simple programming languages which let us load and store values to/from memory and in registers, and to perform arithmetic operations. We will use MIPS assembly language in this course. For example, `add $3, $1, $2` means "add together the values in registers 1 and 2 and place the result in register 3." Note that the destination comes first. Note that most instructions you can use in the course cannot take constant values as arguments!

3.2 Registers

- Registers in MIPS each hold 32 bits of information, and can be thought of as the variables that we are given to work with in our program.
- Some registers are special:
  - \$0 is always 0, and cannot be modified in any ways.
  - \$31 is reserved for the return address. If you lose the value of this register, your program cannot return.
  - We will make \$3, \$29, and \$30 special by convention in this course.

  Try to get in the habit of using them only for their intended purpose, which will explained later on in the course.

3.3 Constant values

Most MIPS instructions in this course deal with registers instead of constant (immediate) values.

We solve this using the load immediate and skip (lis) instruction with the .word directive to load a value into a register. For example,

```
lis $5
.word 7
```

Stores the value 7 into $5.

3.4 Assembling assembly language

The process of turning assembly language into machine code a computer understands is called assembling. Soon we will give you access to an assembler, but for now we'll do it by hand. The MIPS assembly language reference sheet (`https://www.student.cs.uwaterloo.ca/~cs241/mips/mipsref.pdf`) gives a template for how to assemble each 32-bit instruction.

3.5 Assembling an instruction: register format

There are two formats given on the reference sheet for assembling instructions: register and immediate (denoted R and I in the second-last column on the MIPS reference sheet). Suppose we want to assemble the register-format instruction `mult $5, $4`.

- First, we need to convert 5 into a 5-bit binary word: 00101. This is s, as seen on the reference sheet.
- Next, we need to convert 4 into a 5-bit binary word: 00100. This is t
- Next, we simply need to replace the s and t values on the reference sheet for mult with these binary words. Since mult has no d register, 00000 is simply substituted in that position (and has already been substituted for you on the reference sheet). We end up with: 0000 0000 1010 0100 0000 0000 0001 1000
- Finally, we need to rewrite this as hexadecimal. Looking up a table or just remembering the bit patterns we end up with 00A40018.

Alternatively, if we wanted to follow the generic register-format template at the top of the sheet, we need to know the f value (the opcode for mult), which can only be found in the mult instructions template and is 011000

3.6 Exercise

Assemble the following program by first writing out its binary representation, then converting it to a hexadecimal representation, and finally using `cs241.wordasm` to make it executable. What does it do? Run it with `mips.twoints` to verify it behaves as you'd expect.

```
lis $5
.word 7
add $1, $1, $5
sub $1, $5, $0
jr $31
```