LSU EE 4720 Homework 1 Due: 29 January 2018

Problem 1: Follow the instructions for class account setup and for homework workflow. Review the comments in `hw01.s` and look for the area labeled "Problem 1," which has procedure named `unsqw` (unsquish) which initially just has simple placeholder code. Those who want to start before getting a class account the assembler for the entire assignment can be found at `http://www.ece.lsu.edu/ee4720/2018/hw01.s.html`.

Routine `unsqw` is called with two arguments. The first, in register `a0`, holds the address of a C-style string which will be called the *input string*. The second argument, in register `a1`, holds the address of a writeable area of memory, which will be called the *output buffer*.

The input string is an ASCII string that has been compressed using a simplified version of the the Lempel-Ziv LZ77 method. LZ77 compresses text by replacing a substring that has appeared before with a reference to its prior occurrence. The reference consists of the length of the prior occurrence and the distance (number of characters back) of the occurrence. For example, consider:

```
All work and no play makes Johnny a dull boy.
All work and no play makes Johnny a dull boy.
All work and no play makes Johnny a dull boy.
All work and no play makes Johnny a dull boy.
```

The compressed string might look something like this:

```
All work and no play makes Johnny a dull boy.
* Length: 46, Distance: 46 // Replace 46 chars starting 46 characters back.
* Length: 92, Distance: 92 // Replace 92 chars starting 92 characters back.
```

The first line consists of 46 characters (including the end-of-line character). The second line of the compressed string consists of a reference to the first line in the form of a length, 46 characters, and how far back to find the start of the first line, also 46 characters. This makes a copy of the first line. The third line of the compressed string consists of a reference to the first two lines. A reference to a prior occurrence of text can refer to anything that has already appeared, not just whole lines.

The size of the compressed string depends upon the size of a reference. Obviously it would be inefficient to encode a reference such as `* Length: 46, Distance: 46` using regular ASCII text. For this assignment two methods of encoding the reference will be used, *Simple* and *Better*.

In both the Simple and Better methods the characters in the original string must be in the range 1_{16} to $7f_{16}$ (inclusive). Character 0 is used to terminate a string and characters 80_{16} and higher are used to mark the start of a reference.

In the Simple method a reference is a three-byte sequence: $80_{16}, L, D$. The sequence always starts with 80_{16}. The second byte, L, is the length of the sequence and the third byte, D, is the distance, the number of characters back at which the duplicated text can be found. Both L and D are unsigned integers. Let p denote the number of uncompressed characters generated so far. Then reference $80_{16}, L, D$ indicates that the uncompressed characters at positions $p - D$ to $p - D + L - 1$ should be copied to the end of the string of uncompressed characters. Using this method the reference `* Length: 46, Distance: 46` would be encoded as `0x80 0x2e 0x2e`. It is okay for $L - D > 0$.

To help with understanding how the compression works, the compressed string in `hw01.s` includes comments that show the length, distance, and the referenced text. For example, consider the assembly language source code below showing two strings, `uncomp` (uncompressed) and `comp_simple`:

```
uncomp:  # Uncompressed data.
        .ascii "\nAll work and no play makes Johnny a dul"   # Idx:    0 -   39
        .ascii "l boy.\nAll work and no play makes Johnny"   # Idx:   40 -   79
        .ascii " a dull boy.\nAll work and no play makes "   # Idx:   80 -  119
        .ascii "Johnny a dull boy.\nAll work and no play "   # Idx:  120 -  159
        .asciiz "makes Johnny a dull boy.\n"                 # Idx:  160 -  184
comp_simple:  # Compressed data Simple Method.
        .ascii "\nAll work and no play makes Johnny a dul"   # Idx:    0 -   39
        .ascii "l boy."                                      # Idx:   40 -   45
       # Idx:    0 =   46 -   46 =  0x2e -  0x2e.  Len: 139 = 0x8b.
        .byte 0x80 139  46  # "\nAll work and no play makes Johnny a dull boy.\nAll work and
no play makes Johnny a dull boy.\nAll work and no play makes Johnny a dull boy.\n"
.byte 0
# Original: 185 B,  Simple Compressed: 49 B,  Ratio: 0.265
```

The references start with assembler directive `.byte`, other text starts with the directive `.ascii`. There are two comments associated with a reference. The comment above `.byte` shows the starting index, $p - D$, of the text to be copied and the length, L. The comment on the reference line shows the copied text.

The first line of the uncompressed text (`uncomp`) (index 0 to 45) appears literally in the compressed text (`comp_simple`). There is a single reference that indicates $L = 139$ characters should be copied starting at $D = 46$ characters back from position $p = 46$. Notice that this is a case where the text to be copied from overlaps the target text, but that works out well for us.

For more examples search for `comp_simple` in `hw01.s`.

A disadvantage of the Simple method is that the distance is limited to 255 bytes and that the 7 least-significant bits of the first byte (the `0x80`) are unused. The Better method fixes both problems.

Problem continued on the next page.

In the Better method a reference starts with a byte having bit 7 (the most-significant bit) set to 1. There are four cases for a reference, the size of a reference ranges in size from 2 to 4 bytes. Call the first byte of a reference R.

Case 1: $R = 80_{16} = 1000\,0000_2, L, D$. This is identical to the simple case. R is followed by two bytes, the first is the length (L) and the second is distance (D).

Case 2: $R = 10ll\,llll_2, D$. R is followed by one byte (D). The six least-significant bits of R are the length, and D is the distance.

Case 3: $R = c0_{16} = 1100\,0000_2, L, D_h, D_l$. R is followed by three bytes, the first is the length (L), the second, D_h, holds bits 15 to 8 of the distance, and the third, D_l, is the 8 last-significant bits (bits 7 to 0) of the distance.

Case 4: $R = 11ll\,llll_2, D_h, D_l$. R is followed two bytes, the first, D_h, holds bits 15 to 8 of the distance, and the second, D_l, is the 8 last-significant bits (bits 7 to 0) of the distance. The six least-significant bits of R hold the length.

Examples of the cases are shown below, these are taken from the assignment code in `hw01.s`.

```
uncomp:  # Uncompressed data.
        .ascii "[Note: it has been cold cold cold cold!]"    # Idx:    0 -   39
        .ascii "\n======================================="   # Idx:   40 -   79
        .ascii "===============================\nAnother "   # Idx:   80 -  119
        .ascii "frigid Arctic airmass is already pushing"    # Idx:  120 -  159
        .ascii " into the region\nand will provide bitter"   # Idx:  160 -  199
        .ascii "ly cold temperatures. Temperatures will\n"   # Idx:  200 -  239
        .ascii "plunge into the teens and 20s tonight an"    # Idx:  240 -  279
        .ascii "d could be quite similar\nWednesday night"   # Idx:  280 -  319
        .ascii ".\n======================================"   # Idx:  320 -  359
        .ascii "================================\n* TEMPE"   # Idx:  360 -  399
        .ascii "RATURE...Lows will fall into the mid tee"    # Idx:  400 -  439
        .ascii "ns to lower 20s\nalong and north of the I"   # Idx:  440 -  479
        .ascii "-10/12 corridor. South of I-10 lows\nwill"   # Idx:  480 -  519
        .ascii " range from 20 to 25. These temperatures"    # Idx:  520 -  559
        .ascii " will be similar\nWednesday night.\n\n* DUR" # Idx:  560 -  599
        .ascii "ATION...Freezing conditions will likely "    # Idx:  600 -  639
        .ascii "last for 12 to 26\nhours over much of the"   # Idx:  640 -  679
        .ascii " warned area tonight and then 12 to 18\nh"   # Idx:  680 -  719
        .asciiz "ours Wednesday night."                      # Idx:  720 -  740

comp_better:  # Compressed data Better Method.
        .ascii "[Note: it has been cold"                     # Idx:    0 -   22

# Case 2:  # Idx:   18 =   23 -   5 =  0x17 -   0x5.  Len:  15 =  0xf.
        .byte 0x8f           0x05  # " cold cold cold"

        .ascii "!]\n="                                       # Idx:   38 -   41

# Case 1:  # Idx:   41 =   42 -   1 =  0x2a -   0x1.  Len:  69 = 0x45.
        .byte 0x80 0x45      0x01  # "====================================================================="
...
# Case 3  # Idx:   40 =  321 - 281 = 0x141 - 0x119.  Len:  72 = 0x48.
        .byte 0xc0 0x48 0x01 0x19  # "\n======================================================================\n"
...
# Case 4  # Idx:  157 =  613 - 456 = 0x265 - 0x1c8.  Len:   4 =  0x4.
        .byte 0xc4      0x01 0xc8  # "ing "
```

The assignment package includes a program, `hw01-comp.cc` that can be used to encode your own text stings, should you want to.

When the code in `hw01.s` is run a testbench will run routine `unsqw` twice, first on code compressed using the simple method and again on text compressed using the better method.

The output of the testbench starts with the uncompressed text (a known correct copy), followed by the results of running `unsqw` on the two compressed text versions. If there is an error it will show at what index (character position) the error occurs and will show the text before and just beyond this index from the output of `unsqw` and the known correct text. You may need to scroll up to see the beginning of the compressed text (use the arrow keys).

(*a*) Complete `unsqw` so that it uncompresses text compressed using the simple method.

(*b*) Complete `unsqw` so that it uncompresses text compressed using the better method.