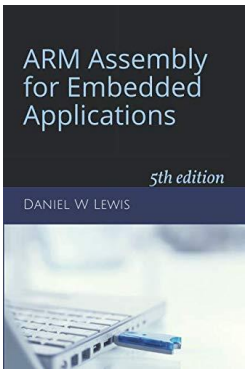


Programming Lab 7F

Huffman Compression

Topics: Bitwise and shift instructions, bit-banding, loops.

Click to download Lab7F-Main.c

Prerequisite Reading: Chapters 1-7

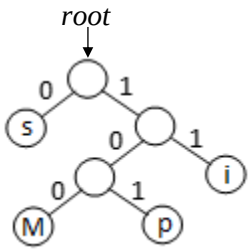
Revised: June 22, 2021

Background*[1]*: In this lab, you are to decode and display a message that has been compressed using Huffman coding. Each character in the message is represented by a unique substring of bits. The code is optimized so that more common characters are represented using fewer bits than less common characters. The substrings are concatenated to form one long string. For example, the word "Mississippi" could be represented as 10011001100111011011, where substrings translate into letters according to the table on the right.

Sub-string	Let-ter
0	s
100	M
101	p
11	i

In facilitate decoding, the table is converted into a binary tree stored as an array of bytes. The byte representing a leaf node contains an ASCII character, while the bytes of all interior nodes contain the integer 0. The root node's byte is stored at array index 0. In general, if the byte index of a node is k, the byte index of its left child is $2k + 1$ and that of its right child is $2k + 2$.

The coded message is decoded and printed using the following algorithm:




```
start:  k ← 0 ;                    // Start at the root of the tree
top:    bit ← GetBit ;             // Get the next bit of the coded message
        k ← 2 × k + 1 + bit ;      // If bit = 0 descend left, else descend right
        byte ← array[k] ;          // Get the content of the node
        if byte = 0 goto top ;     // If it's zero, it's an interior node
        if byte = '$' return ;     // If it's a dollar sign, you're done
        Display1Char(byte);        // Otherwise it's a leaf node: print the character
        goto start ;               // Go back to top of tree and decode next character
```

Assignment: The main program will compile and run without writing any assembly. However, your task is to create equivalent replacements in assembly language for the following two functions found in the C main program. The original C versions have been defined as "weak" so that the linker will automatically replace them in the executable image by those you create in assembly; you do not need to remove the C versions. This allows you to create and test your assembly language functions one at a time. The parameter `msg` is a pointer to the bits of the coded message packed 8 per byte and `array` holds the binary decoding tree. The first bit of the message is the least-significant bit of the first byte of memory pointed to by parameter `msg`.

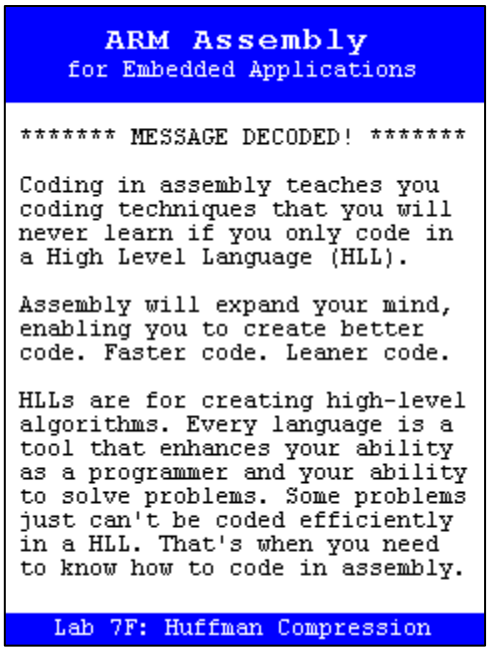
```
void DecodeMessage(void *msg, char array[]) ;
```

Note that function `DecodeMessage` must call C function `Display1Char` that is provided in the main program; do not recreate this function in assembly. Your assembly language code will likely need to push and pop registers; be sure that the total number of registers you push and pop is even so that the address held in the stack pointer remains a multiple of eight to satisfy the data alignment requirements of compiled C code.

You will also need to implement a second function in assembly. Function `GetBit` returns either 0 or 1, corresponding to a bit of the coded message located at position `bitnum` (starting at position 0).

```
int GetBit(void *msg, uint32_t bitnum) ;
```

Implement `GetBit` two ways: (1) use bitwise and shift operations, and (2) use Bit-Banding. Test your solution using the C main program.



[1] https://en.wikipedia.org/wiki/Huffman_coding