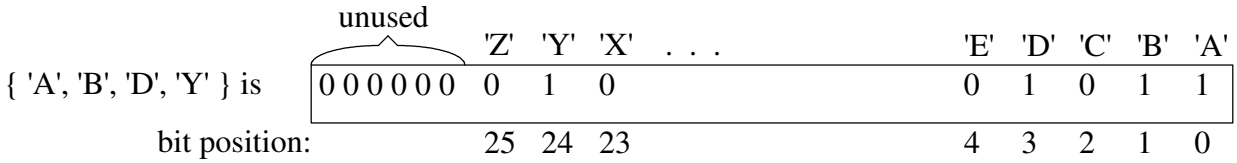
Comp Org HW #8 Due: May 4 (Sat) 11:59 PM

Bit-string Set of Letters: (Here we are completing the operations started in Lecture 23**. You can find code for the bitString and Union there.)**

You are to complete the operations for the set of letters using a bit string. Recall, the bit string representation for the set of letters can use a 32-bit word with the least-significant bit associated with the letter 'A', etc.



The set of letters should have the following operations (subprograms):

Subprogram Name	**Parameters**	**Description**
bitString (done in Lecture 23)	▪pass in a pointer to the an .ASCIIZ string ▪returns a word containing the set of letters as a bitString	Returns a bit string corresponding to the set of letters in the .ASCIIZ string. Non-letter characters are ignored, and both upper and lower-case letters should be represented as the upper-case letter.
union (done in Lecture 23)	▪passed two set bitStrings ▪returns the set union of the two sets	The resulting set should contain the elements that are in one or both of the input sets.
intersection	▪passed two set bitStrings ▪returns the set intersection of the two sets	The resulting set should contain the elements that are in both of the input sets.
difference	▪passed two set bitStrings ▪returns the set difference of the first set - second set	The resulting set should contain the elements that are in the first set, but not also in the second set.
contains	▪passed an .ASCII character and a set bitString ▪returns a Boolean (0 for false or 1 for true)	Returns 1 (true) if the .ASCII character is in the bitString set; otherwise return 0 (false).
print	▪passed an set bitString	Prints the bitString to the console using the print_string system call. The set should be printed in the conventional format, i.e., "{ E, G, T, Y }"

Additionally, you should have a main program that

1) allows a user to interactively enter two strings (use the PCSpim I/O syscall),
2) constructs two bitString sets from these strings,
3) prints the set of letters contained in each string,
4) determines and prints the union, intersection, and difference of the two bitString sets from (1) and (2),
5) checks to see if the first bitString set contains the letters: 'A', 'Y', and 'Z'. The results of each of these checks should be printed to the console.

You should submit your homework via the Internet by following the directions at:

http://www.cs.uni.edu/~fienup/cs1410s19/homework/submissionDirections.htm

Basically, you put the file hw8.s in a `hw8` folder and zip the folder to create a `hw8.zip` file containing:

- the MIPS assembly language program, e.g., hw8.s from any text-editor (e.g., WordPad),
- a window capture of the **output window after running your assembly language program using the two strings: "Bats and balls" and "BIGGER IS BETTER"**

```
# Partial code to implement a bit-string of letters
       .data
str1:  .asciiz "Cape3?!AE"
str2:  .asciiz "A d y B**#&."
set1:  .word 0
set2:  .word 0

       .text
       .globl main
main:
              la $a0, str1
              jal bitString
              sw $v0, set1

              la $a0, str2
              jal bitString
              sw $v0, set2

              li $v0, 10
              syscall
bitString:
# bitString Algorithm:
# resultSet = {}
# index = 0
# while True:
#    nextChar = str[index]
#    if nextChar == 0 then   // the NULL character
#        break
#    end if
#    if nextChar >= ascii of 'a' and nextChar <= ascii of 'z' then
#         convert it upper-case letter by subtracting 32
#    end if
#    if nextChar >= ascii of 'A' and nextChar <= ascii of 'Z' then
#        resultSet = resultSet U {nextChar}
#    end if  (no else because we are ignoring non-letters)
#       index = index + 1
# end while
# return resultSet

# Register Usage - NOTE: doesn't call anything so by using only $a and $t registers, doesn't need
#                        to save on stack
# $a0 parameter contains address of .asciiz string, but will be walked down the string
# $v0 used for the resultSet
# $t0 used to hold nextChar ASCII value
# $t3 used to hold the mask for the str[index] character
       li $v0, 0   # resultSet = {}
while:
       lb $t0, 0($a0)
       beq $t0, 0, end_while  # NULL character (0) detected at end of .asciiz
if_1:  blt $t0, 97, end_if_1  # ASCII for 'a' is 97
       bgt $t0, 122, end_if_1 # ASCII for 'z' is 122
       addi $t0, $t0, -32     # convert to upper-case letter
end_if_1:
if_2:  blt $t0, 65, end_if_2      # ASCII for 'A' is 65
       bgt $t0, 90, end_if_2      # ASCII for 'Z' is 90
       addi $t8, $t0, -65         # determine bit position of letter in bit-string
       li $t3, 1                  # Build mask: start with 1 at right-most position
       sllv $t3, $t3, $t8         # Build mask: move 1 to correct position to finish building mask
       or $v0, $v0, $t3           # update resultSet in $v0 = $v0 bit-wise-OR with mask
end_if_2:
       addi $a0, $a0, 1    # walk-pointer to str[index] to next character
       j while
end_while:
       jr $ra
```