COL216 Assignment 1

By: Rajat Mahaur 2020CS50534

Stage:3 Report

The folder contains four files one is Stage3Main.s and CompareString.s and CompareStringModified.s and MergeSort.s

1 Stage3Main.s

The .extern comparesortlist helps to call the comparesortlist label from the MergeSort.s and pass the arguments. The .global fs, size, firststringadd, duplicon serve as input providers and some other purpose for MergeSort.s. It provides input as-
1. r4: The starting address of the list containing the addresses of the strings in first sorted string list and may also be from firststringadd
Note: r5 provides the Condition to determine if wanted the case-sensitive or case-insensitive. Number of strings in unsorted list is provided in "size" memory value which is fed in different registers at different times. "duplicon" has condition to determine if wanted a list to contain duplicate strings or not is received as another input.

The program's main label take input for the Condition to determine if wanted the case-sensitive or case-insensitive and put the value in r5 (0 for non case sensitive and 1 for case sensitive) and Condition to determine if wanted a list to contain duplicate strings or not(0 for non case duplicacy and 1 for duplicacy) and pushed it into the stack. Later in program, I take the input of the characters in linear fashion in "fs" space, different sorted strings are entered in one go seperated by a whitespace and when the assembler encounters a space character as input it stores the address of ahead byte into "firststringadd" which contains the array of starting address of string. When the newline character is encountered that is the time when user has finished entering the first string list and replace it with null character in "fs" thus forming "fs" a continous array of string characters each string ceperated by null character, the loop is exited in this condition.
The branch link is done to call comparesortlist label and it returns the result that is address of the list containing the sorted list in r4, and the number of final strings in this in r8.
Later the program runs "test" and "prt" loop, "prt" nested in "test", where

"test" loops over every string in merged list conditioning over r8 which is decremented by 1 every time until it goes over every string. The "prt" prints the characters in every string in this merged list seperated by whitespaces.
Program ends.

2 MergeSort.s

The .extern comparelist helps to call the comparelist label from the CompareStringModified.s and pass the arguments to it and .global comparesortlist helps to make the comparesortlist label visible to Stage3Main.s.

The program extracts the input fed to in from the stack and store them as following-
1. r4: The starting address of the list containing the addresses of the strings in first sorted string list and may also be from firststringadd
Note: r5 receives the Condition to determine if wanted the case-sensitive or case-insensitive. Number of strings in unsorted list is received in "size" memory value which is fed in different registers at different times. "duplicon" has condition to determine if wanted a list to contain duplicate strings or not is received as another input. It first stores(in putstart and loop1 label) the value of lr in "link" as lr content will be changed once called comparelist, then it saves number 1(accounting to strings containing only single strings) followed by all the address of firststringadd in storestarts thus mark the point of dividing the input unsorted list in individual strings. "savehere" is assigned the ending address of the place where these individual strings are done entering for new srted list containing any combinations of these individual strings will saved after loop2.
Now in loop2 label, placenew is assigned address of storestarts.
then in loop2, I recursively call comparelist label on srted list, pointer(r2,r9) over storestarts at one time of loop push any two string List sorted(individual strings are always sorted) into stack as this serve as the input for CompareStringModified.s, the input are pushed in folllowing order-
1. Condition to determine if wanted a list to contain duplicate strings or not
2. The number of strings entered in first sorted string list
3. The starting address of the list containing the addresses of the strings in first sorted string list
4. The number of strings entered in second sorted string list
5. The starting address of the list containing the addresses of the strings in second sorted string list
The CompareStringModified.s merge these sorted list and thus finally the r4 contains the address of the list containing the merged sorted strings and r8 and stack contains the total number of strings in this returned merged list.
Again these number of strings and pointer to merged list is save at the address pointed by savehere.
The pointer over the storestarts keep on scanning storestarrs and push a couple of sorted string list in CompareStringModified.s and loop2 thus will run until

the total number of string returned in sorted merged list is equal to the total number of strings in input unsorted list as that this point there is a list which is having all strings and they are sorted.
Now it retrieve the condition of duplicacy from "duplicon" and if it says 1(ie duplicated string list is required) then it put the total number of strings (size memory value) in r8 and address of list containing start addresses of the string in sorted order in r4 and branch to "backtomain" label.
Else if it is 0(ie remove duplicates) then it runs over "loopfinal" labelled loop which compare the adjacent strings of the list received by compare label and store the list into "finalnon" if it is different else skips over it and checks for other. In this way finalnon contains non-duplicated strings. At last the value of the total number of list(ie value in stack in continuously loaded into r8 and decremented each time by 1 if strings are equal else not and again push into stack.
Thus finally the r4 contains the address of the list containing the merged sorted strings and r8 contains the total number of strings in this returned merged list, which is returned to main by copyig the content of "link" into pc and thus back to main program for printing it.

3 CompareStringModified.s

The .extern compare helps to call the compare label from the CompareString.s and pass the arguments to it and .global comparelist helps to make the comparelist label visible to Stage2Main.s.

The program extracts the input fed to in from the stack and store them as following-
2. r4: The number of strings entered in first sorted string list
3. r10: The starting address of the list containing the addresses of the strings in first sorted string list
4. r7: The number of strings entered in second sorted string list
5. r11: The starting address of the list containing the addresses of the strings in second sorted string list
Note: r5 receives the Condition to determine if wanted the case-sensitive or case-insensitive.

r12 contains the address of list "final" assigned by pointstar which our dummy list to contain the starting address of all sorted strings in merged from. lr is pushed into stack as branch link is done to CompareString.s . the loop runs over the both adddress of unsorted list recieved at a time and at a time sends the addresses of the string to compare in r2 and r3 while condition of case in r5 to the compare label in CompareString.s and gets the result based on comparision of the strings in r6. As comparing this value to the short string address is stored into r12 and the address of the other string is remained same while the string pushed acquires the new address by incremented respective list address. Now

with the inserting I decrement the number of string by 1 as a string address is stored in r12 and check the number of string in the both, as soon as one of the register value(r4 or r7) equals zero in this case one string is fully inserted into r12, I break loop and store the other list left strings into final by help of other loops(loopnew1 or loopnew2).
"pointstar" is again assigned back to the place were new merger string has ended getting entered in "final", so that next time it will the place to enter new merged sorted list.
Now addition of the numbers of strings in both the list is pushed from stack and store their addition in stack as well.
Thus finally the r4 contains the address of the list containing the merged sorted strings and r8 and stack contains the total number of strings in this returned merged list.

4 CompareString.s

Note: It's working is same as in Stage 1 and copy pasting the entire working here from stage 1 report.
compare label of this file is decared global so as to be called by the CompareStringList.s The inputs received in r5 is compared with 0 and if equal is set then branch to nonconse label. compare label checks for case sensivitive mode based on the ascii value of the strings, the loop takes and read the values of address from r2 and r3 and while incrementing then it checks wether greater or less(in this case the r6 is set to 1 or -1 respectively and return back to CompareStringList.s and give 1 output) or if they are equal then the loop continues while checking any one of the characters in comparing is not null, if null then the r6 is set equal to 0 which the case that the bot strings are equal and output is given in Stage1Main.s according r6 value.
nonconse label also works same as the compare label but the difference is that firstly before comparing the characters it convert both the characters read from each strings in lowercase but comparing their ascii value is less than 91 and then store results in r8(for r2 addressed string) and r9(for r3 addressed string) and now comapre the characters in r8 and r9 and update the r6, the working is same as of compare label. The output is then given in CompareStringList.s according r6 value.

5 Data:

1. fs: Store the input string unsorted character by character where null character lies after every string.
2. size: Number of strings in input unsorted list.
3. firststringadd: List containing strating addresses of the strings in fs.
4. duplicon: Saves the duplication condition.
5. dummychar: A dummy character for input of characters while taking input

of strings and others.
6. storestarts: Store number of strings in the list followed by the address of the string list containing the starting address of unsorted strings.
7. pointstar: store the point address of final from where the new sorted string list will be saved.
8. final: String List containing starting address of strings in sorted manner.
9. savehere: store the address in storestarts where, the number of strings followed by the address of the list containing starting address of the sorted strings formed by merging two unsorted string list, is saved in storestarts.
10: placenew: is the pointer to the address in storestarts where it needs to start reading the storestarts to pus stuff into stack.
11. finalnon: Incase when wanted non duplicated string list this stores the list pointers which are not duplicated.

6 Assumptions

1. I am using Angel SWI Instructions in this.
2. I am using ARMSim201 for this assignment.
3. Note that whatever combination of strings you are choosing I expect you to assume order of strings in sorted manner in way:
if you are choosing for finding the case insensitive case, then
"Rajat" "meena" are not sorted as case insensitively you are making "R" greater than "m" case insensitively
while "meena" "Rajat" are sorted
4. Note that if you are asking for non duplicate strings among case insensitive then: "RAJAT" "Rajat" "rajat" are all duplicated according to my program and thus any one of them will be returned in sorted list.

7 Input Output test cases format

TEST CASE 1
Enter the mode 0 for non case sensitive and 1 for case sensitive
0
Enter the mode 0 for non case duplicacy and 1 for duplicacy
0
Enter the string list seperated by whitespace in between each string and press enter when finished entering
Rajat rajat gauri Gau RAJAT
The final merged sorted string list is
Gau gauri rajat

TEST CASE 2
Enter the mode 0 for non case sensitive and 1 for case sensitive
0

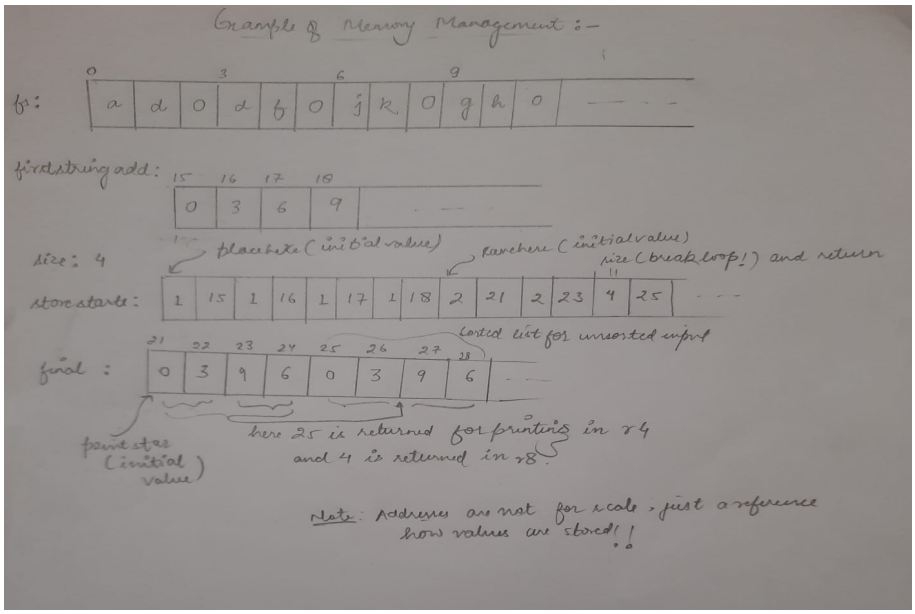
Enter the mode 0 for non case duplicacy and 1 for duplicacy
1
Enter the string list seperated by whitespace in between each string and press enter when finished entering
Rajat rajat gauri Gau RAJAT
The final merged sorted string list is
Gau gauri rajat Rajat RAJAT

TEST CASE 3
Enter the mode 0 for non case sensitive and 1 for case sensitive
1
Enter the mode 0 for non case duplicacy and 1 for duplicacy
0
Enter the string list seperated by whitespace in between each string and press enter when finished entering
Rajat Rajat rajat RAJAT Gau gauri
The final merged sorted string list is
Gau RAJAT Rajat gauri rajat

TEST CASE 4
Enter the mode 0 for non case sensitive and 1 for case sensitive
1
Enter the mode 0 for non case duplicacy and 1 for duplicacy
1
Enter the string list seperated by whitespace in between each string and press enter when finished entering
Rajat Rajat rajat RAJAT Gau gauri
The final merged sorted string list is
Gau RAJAT Rajat Rajat gauri rajat

8 Memory Management Example

This example is not for scale for memory, it is just a reference.



9 Additional points

Comments are also added in all four files to understand better.
The all three program files are solely work of mine and whole coding is done by me. The help regarding the syntaxes of commands and some information about ARM is taken either from books or google.