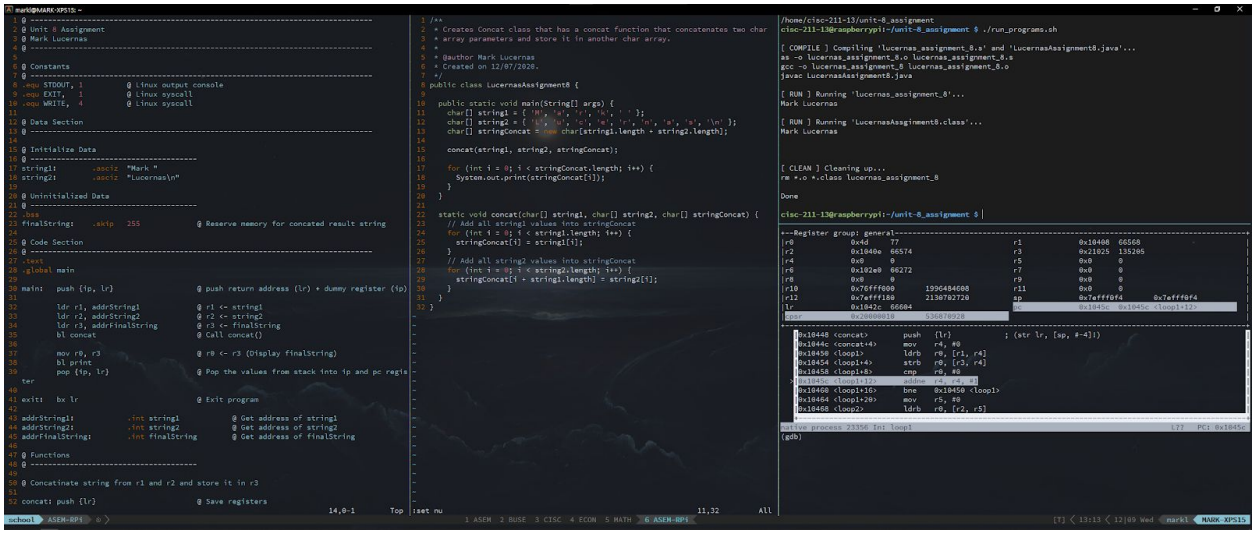
Mark Lucernas
CISC 211
Prof. Saied Moezzi
December 9, 2020

cisc-211-13@raspberrypi

Unit 8 Assignment

Problem 1

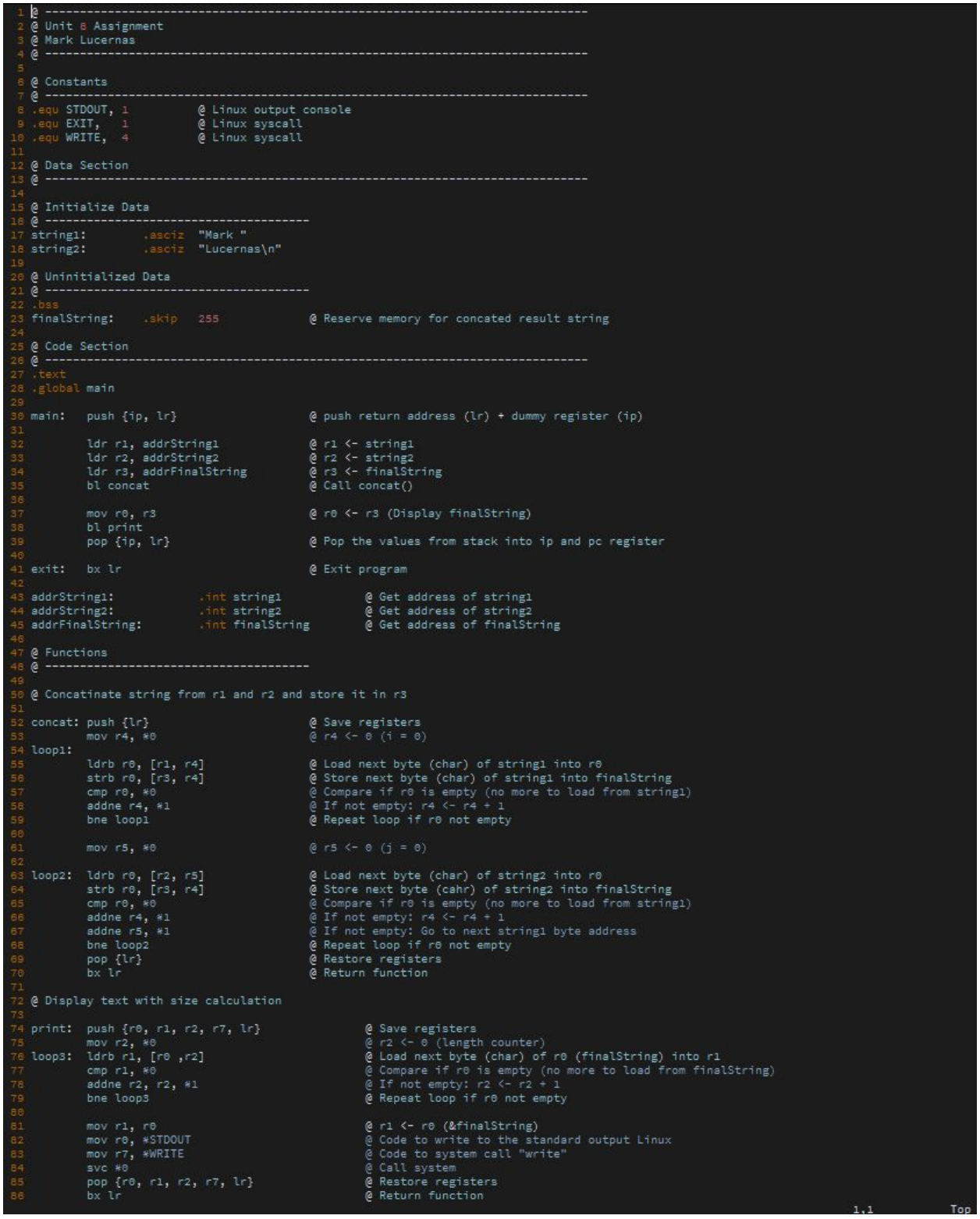
Overview



Compile and Run

```
cisc-211-13@raspberrypi:~/unit-8_assignment $ pwd
/home/cisc-211-13/unit-8_assignment
cisc-211-13@raspberrypi:~/unit-8_assignment $ ./run_programs.sh

[ COMPILE ] Compiling 'lucernas_assignment_8.s' and 'LucernasAssignment8.java'...
as -o lucernas_assignment_8.o lucernas_assignment_8.s
gcc -o lucernas_assignment_8 lucernas_assignment_8.o
javac LucernasAssignment8.java

[ RUN ] Running 'lucernas_assignment_8'...
Mark Lucernas

[ RUN ] Running 'LucernasAssginment8.class'...
Mark Lucernas



[ CLEAN ] Cleaning up...
rm *.o *.class lucernas_assignment_8

Done

cisc-211-13@raspberrypi:~/unit-8_assignment $
```

Assembly Screenshot



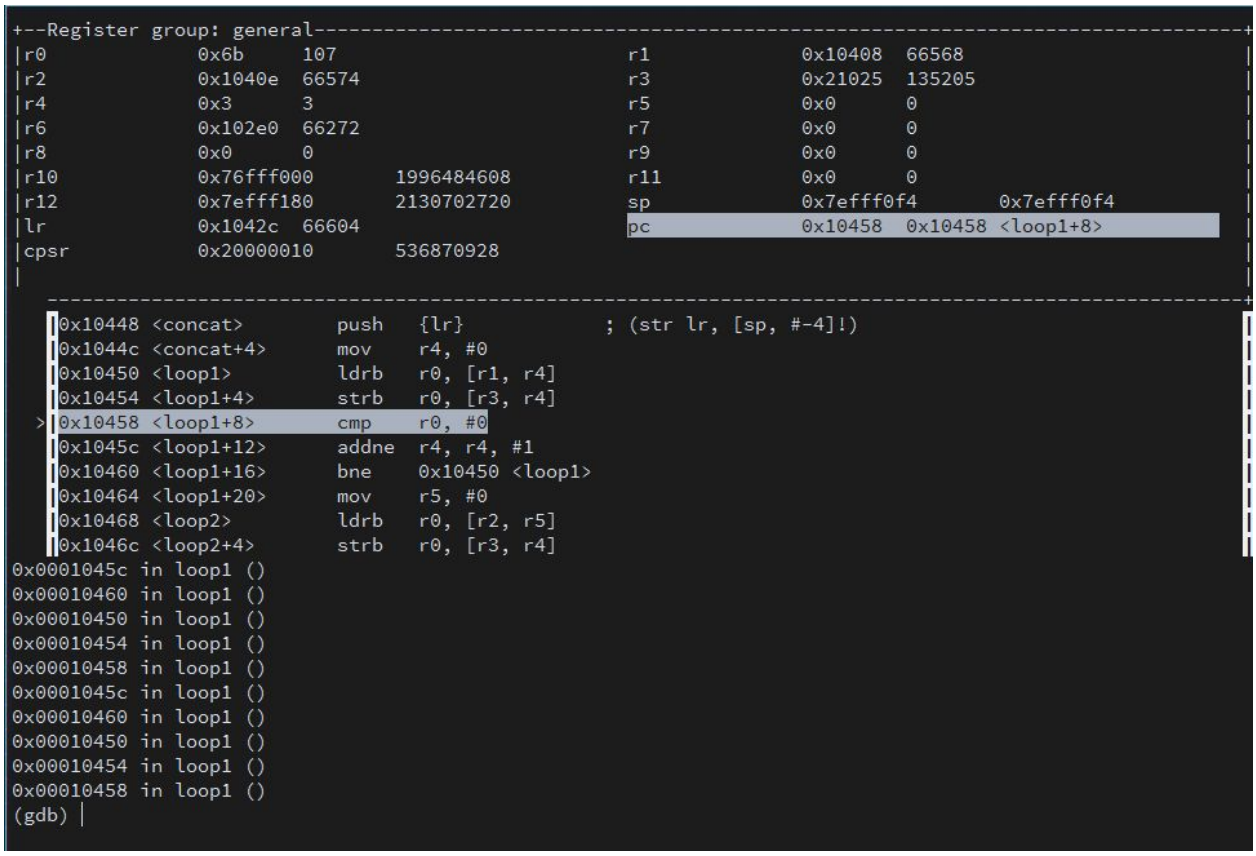
```
@ ------------------------------------------------------------------------------
@ Unit 6 Assignment
@ Mark Lucernas
@ ------------------------------------------------------------------------------

@ Constants
@ ------------------------------------------------------------------------------
.equ STDOUT, 1          @ Linux output console
.equ EXIT,   1          @ Linux syscall
.equ WRITE,  4          @ Linux syscall

@ Data Section
@ ------------------------------------------------------------------------------

@ Initialize Data
@ --------------------------------------
string1:        .asciz  "Mark "
string2:        .asciz  "Lucernas\n"

@ Uninitialized Data
@ --------------------------------------
.bss
finalString:    .skip   255             @ Reserve memory for concated result string

@ Code Section
@ ------------------------------------------------------------------------------
.text
.global main

main:   push {ip, lr}                   @ push return address (lr) + dummy register (ip)

        ldr r1, addrString1             @ r1 <- string1
        ldr r2, addrString2             @ r2 <- string2
        ldr r3, addrFinalString         @ r3 <- finalString
        bl concat                       @ Call concat()

        mov r0, r3                      @ r0 <- r3 (Display finalString)
        bl print
        pop {ip, lr}                    @ Pop the values from stack into ip and pc register

exit:   bx lr                           @ Exit program

addrString1:            .int string1            @ Get address of string1
addrString2:            .int string2            @ Get address of string2
addrFinalString:        .int finalString        @ Get address of finalString

@ Functions
@ --------------------------------------

@ Concatinate string from r1 and r2 and store it in r3

concat: push {lr}                       @ Save registers
        mov r4, #0                      @ r4 <- 0 (i = 0)
loop1:
        ldrb r0, [r1, r4]               @ Load next byte (char) of string1 into r0
        strb r0, [r3, r4]               @ Store next byte (char) of string1 into finalString
        cmp r0, #0                      @ Compare if r0 is empty (no more to load from string1)
        addne r4, #1                    @ If not empty: r4 <- r4 + 1
        bne loop1                       @ Repeat loop if r0 not empty

        mov r5, #0                      @ r5 <- 0 (j = 0)

loop2:  ldrb r0, [r2, r5]               @ Load next byte (char) of string2 into r0
        strb r0, [r3, r4]               @ Store next byte (cahr) of string2 into finalString
        cmp r0, #0                      @ Compare if r0 is empty (no more to load from string1)
        addne r4, #1                    @ If not empty: r4 <- r4 + 1
        addne r5, #1                    @ If not empty: Go to next string1 byte address
        bne loop2                       @ Repeat loop if r0 not empty
        pop {lr}                        @ Restore registers
        bx lr                           @ Return function

@ Display text with size calculation

print:  push {r0, r1, r2, r7, lr}               @ Save registers
        mov r2, #0                              @ r2 <- 0 (length counter)
loop3:  ldrb r1, [r0 ,r2]                       @ Load next byte (char) of r0 (finalString) into r1
        cmp r1, #0                              @ Compare if r0 is empty (no more to load from finalString)
        addne r2, r2, #1                        @ If not empty: r2 <- r2 + 1
        bne loop3                               @ Repeat loop if r0 not empty

        mov r1, r0                              @ r1 <- r0 (&finalString)
        mov r0, #STDOUT                         @ Code to write to the standard output Linux
        mov r7, #WRITE                          @ Code to system call "write"
        svc #0                                  @ Call system
        pop {r0, r1, r2, r7, lr}                @ Restore registers
        bx lr                                   @ Return function
                                                                        1,1           Top
```

Assembly Source Code

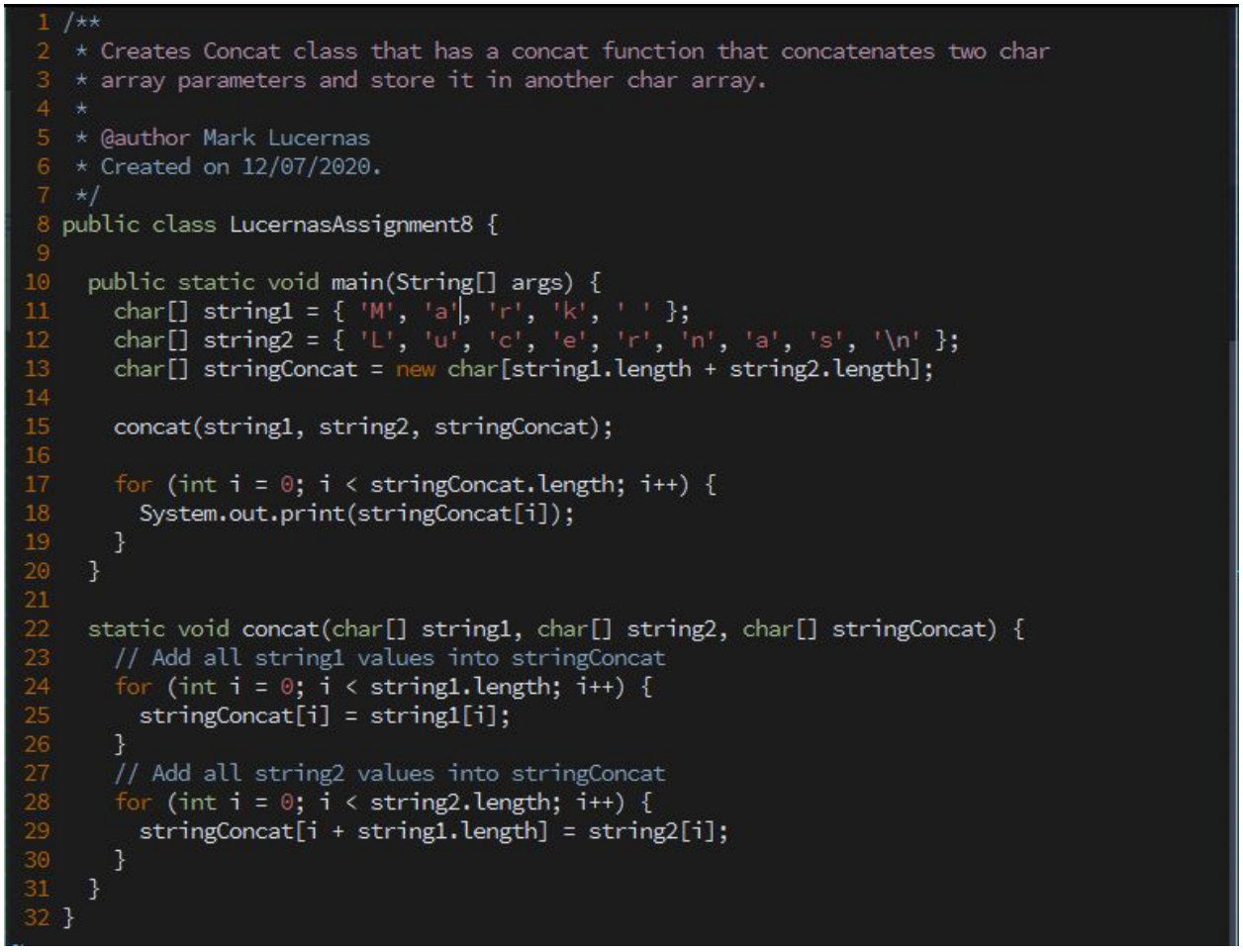
```
@ ------------------------------------------------------------------------------
@ Unit 8 Assignment
@ Mark Lucernas
@ ------------------------------------------------------------------------------

@ Constants
@ ------------------------------------------------------------------------------
.equ STDOUT, 1                  @ Linux output console
.equ EXIT,   1                  @ Linux syscall
.equ WRITE,  4                  @ Linux syscall

@ Data Section
@ ------------------------------------------------------------------------------

@ Initialize Data
@ ---------------------------------------
string1:            .asciz  "Mark "
string2:            .asciz  "Lucernas\n"

@ Uninitialized Data
@ ---------------------------------------
.bss
finalString:    .skip   255                 @ Reserve memory for concated result string

@ Code Section
@ ------------------------------------------------------------------------------
.text
.global main

main:   push {ip, lr}                       @ push return address (lr) + dummy register (ip)

        ldr r1, addrString1                 @ r1 <- string1
        ldr r2, addrString2                 @ r2 <- string2
        ldr r3, addrFinalString             @ r3 <- finalString
        bl concat                           @ Call concat()

        mov r0, r3                          @ r0 <- r3 (Display finalString)
        bl print
        pop {ip, lr}                        @ Pop the values from stack into ip and pc register

exit:   bx lr                               @ Exit program

addrString1:        .int string1        @ Get address of string1
addrString2:        .int string2        @ Get address of string2
addrFinalString:    .int finalString    @ Get address of finalString
```

```
@ Functions
@ ------------------------------------

@ Concatinate string from r1 and r2 and store it in r3

concat: push {lr}                       @ Save registers
        mov r4, #0                      @ r4 <- 0 (i = 0)
loop1:
        ldrb r0, [r1, r4]               @ Load next byte (char) of string1 into r0
        strb r0, [r3, r4]               @ Store next byte (char) of string1 into finalString
        cmp r0, #0                      @ Compare if r0 is empty (no more to load from string1)
        addne r4, #1                    @ If not empty: r4 <- r4 + 1
        bne loop1                       @ Repeat loop if r0 not empty

        mov r5, #0                      @ r5 <- 0 (j = 0)

loop2:  ldrb r0, [r2, r5]               @ Load next byte (char) of string2 into r0
        strb r0, [r3, r4]               @ Store next byte (cahr) of string2 into finalString
        cmp r0, #0                      @ Compare if r0 is empty (no more to load from string1)
        addne r4, #1                    @ If not empty: r4 <- r4 + 1
        addne r5, #1                    @ If not empty: Go to next string1 byte address
        bne loop2                       @ Repeat loop if r0 not empty
        pop {lr}                        @ Restore registers
        bx lr                           @ Return function

@ Display text with size calculation

print:  push {r0, r1, r2, r7, lr}       @ Save registers
        mov r2, #0                      @ r2 <- 0 (length counter)
loop3:  ldrb r1, [r0 ,r2]               @ Load next byte (char) of r0 (finalString) into r1
        cmp r1, #0                      @ Compare if r0 is empty (no more to load from finalString)
        addne r2, r2, #1                @ If not empty: r2 <- r2 + 1
        bne loop3                       @ Repeat loop if r0 not empty

        mov r1, r0                      @ r1 <- r0 (&finalString)
        mov r0, #STDOUT                 @ Code to write to the standard output Linux
        mov r7, #WRITE                  @ Code to system call "write"
        svc #0                                  @ Call system
        pop {r0, r1, r2, r7, lr}        @ Restore registers
        bx lr                           @ Return function
```

GDB Debugger of Assembly Code



High-level Language: Java Screenshot



```java
/**
 * Creates Concat class that has a concat function that concatenates two char
 * array parameters and store it in another char array.
 *
 * @author Mark Lucernas
 * Created on 12/07/2020.
 */
public class LucernasAssignment8 {

  public static void main(String[] args) {
    char[] string1 = { 'M', 'a', 'r', 'k', ' ' };
    char[] string2 = { 'L', 'u', 'c', 'e', 'r', 'n', 'a', 's', '\n' };
    char[] stringConcat = new char[string1.length + string2.length];

    concat(string1, string2, stringConcat);

    for (int i = 0; i < stringConcat.length; i++) {
      System.out.print(stringConcat[i]);
    }
  }

  static void concat(char[] string1, char[] string2, char[] stringConcat) {
    // Add all string1 values into stringConcat
    for (int i = 0; i < string1.length; i++) {
      stringConcat[i] = string1[i];
    }
    // Add all string2 values into stringConcat
    for (int i = 0; i < string2.length; i++) {
      stringConcat[i + string1.length] = string2[i];
    }
  }
}
```

High-level Language: Java Source Code

```java
/**
 * Creates Concat class that has a concat function that concatenates two char
 * array parameters and store it in another char array.
 *
 * @author Mark Lucernas
 * Created on 12/07/2020.
 */
public class LucernasAssignment8 {

  public static void main(String[] args) {
    char[] string1 = { 'M', 'a', 'r', 'k', ' ' };
    char[] string2 = { 'L', 'u', 'c', 'e', 'r', 'n', 'a', 's', '\n' };
    char[] stringConcat = new char[string1.length + string2.length];
```

```
    concat(string1, string2, stringConcat);

    for (int i = 0; i < stringConcat.length; i++) {
      System.out.print(stringConcat[i]);
    }
  }

  static void concat(char[] string1, char[] string2, char[] stringConcat) {
    // Add all string1 values into stringConcat
    for (int i = 0; i < string1.length; i++) {
      stringConcat[i] = string1[i];
    }
    // Add all string2 values into stringConcat
    for (int i = 0; i < string2.length; i++) {
      stringConcat[i + string1.length] = string2[i];
    }
  }
}
```

Problem 2

Exercise 7.2

a) **STR** and **B**. because these instructions write to the register file when they shouldn't.