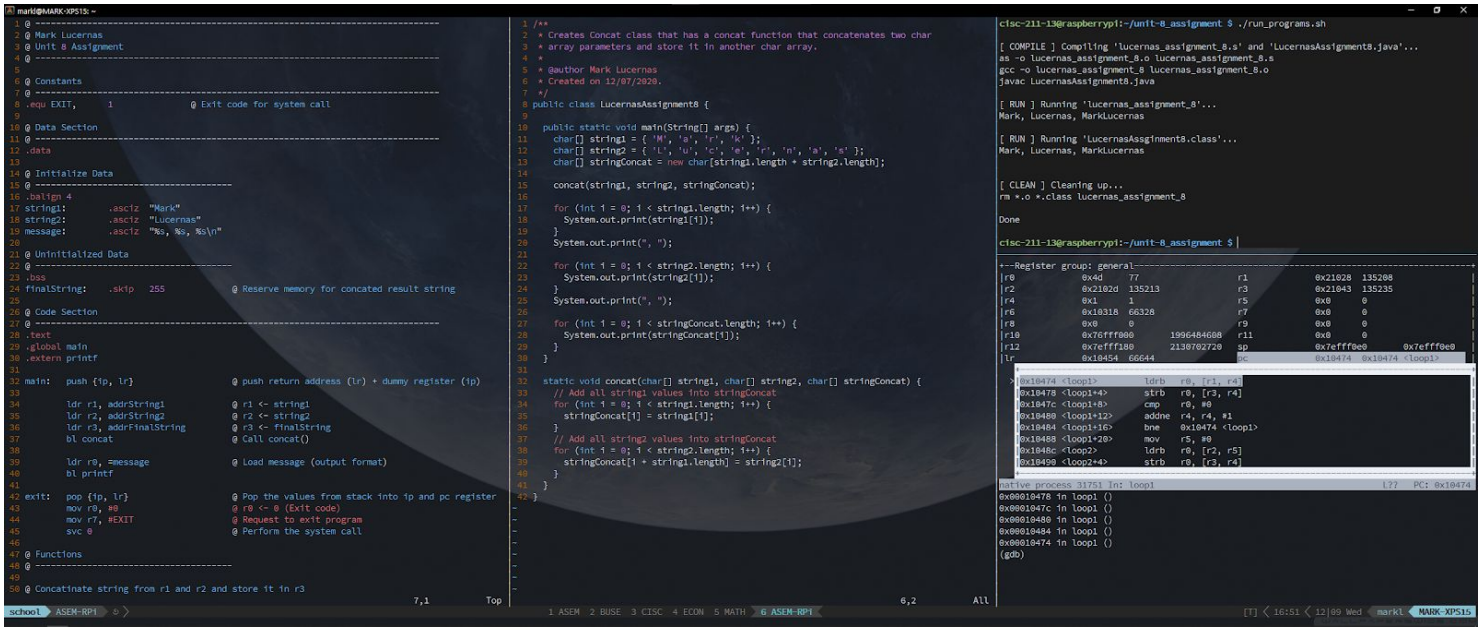
Mark Lucernas
CISC 211
Prof. Saied Moezzi
December 9, 2020

cisc-211-13@raspberrypi

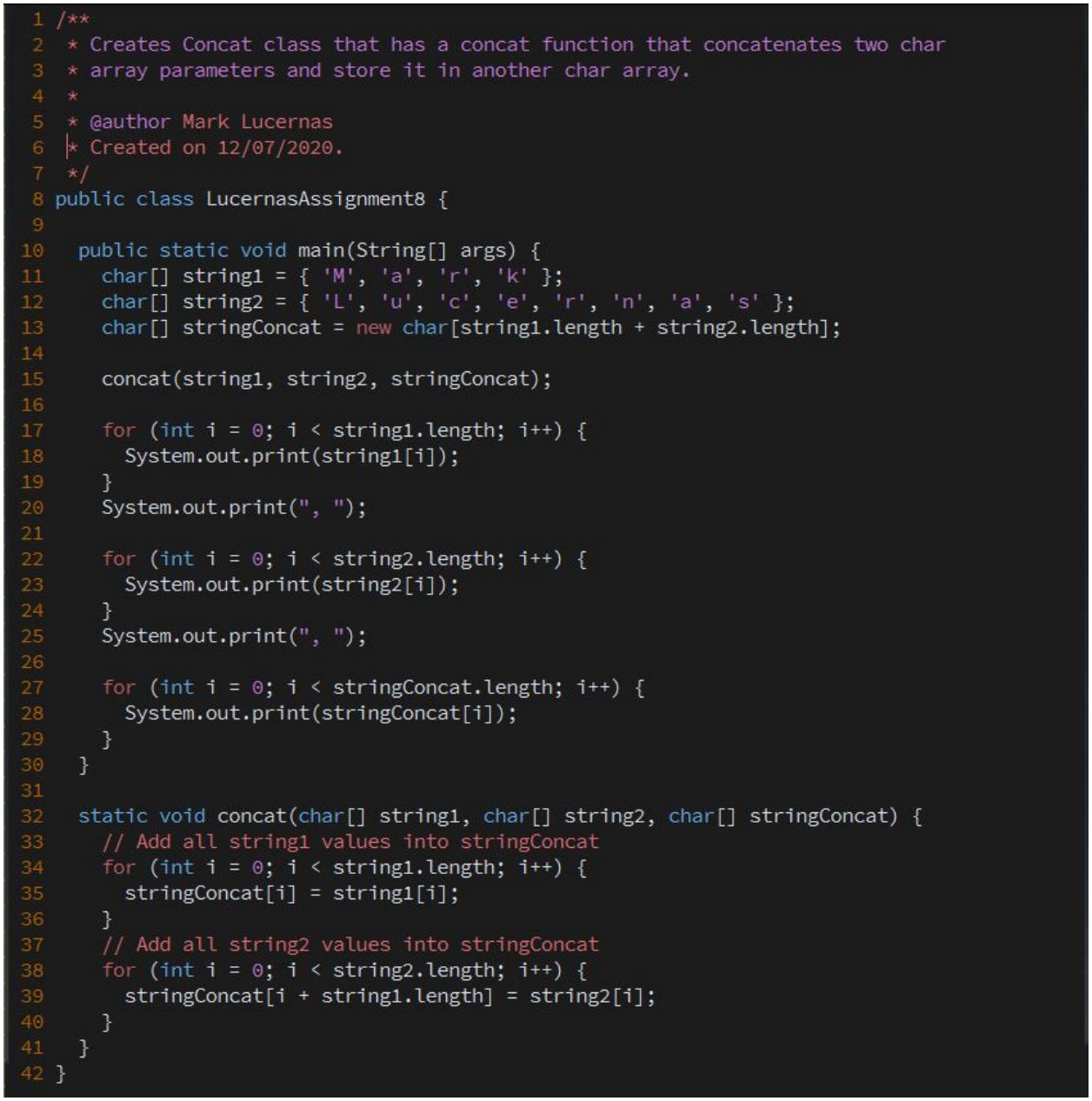
# Unit 8 Assignment

## Problem 1

### Overview



## High-level Language: Java Screenshot



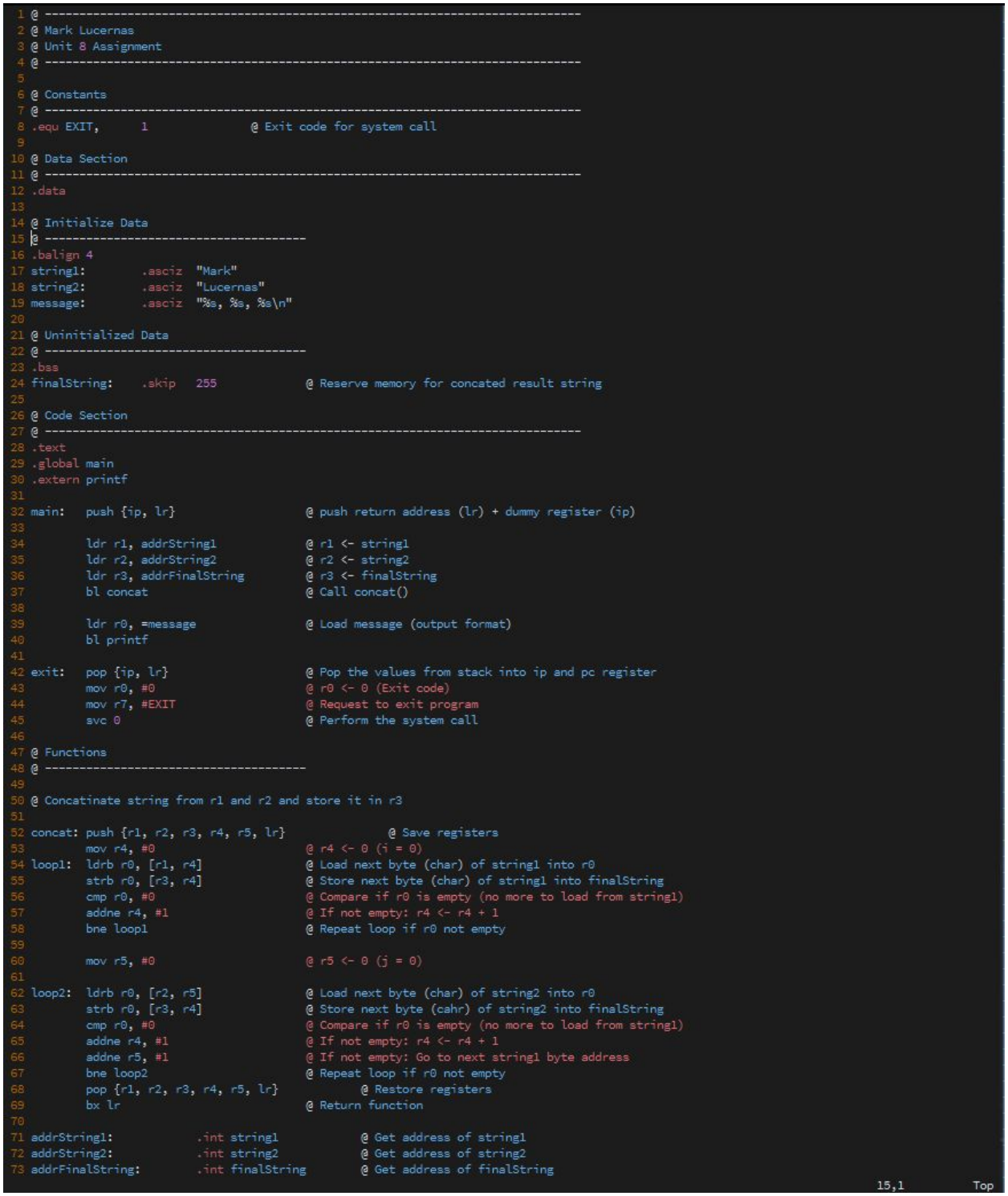
```java
/**
 * Creates Concat class that has a concat function that concatenates two char
 * array parameters and store it in another char array.
 *
 * @author Mark Lucernas
 * Created on 12/07/2020.
 */
public class LucernasAssignment8 {

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

    concat(string1, string2, stringConcat);

    for (int i = 0; i < string1.length; i++) {
      System.out.print(string1[i]);
    }
    System.out.print(", ");

    for (int i = 0; i < string2.length; i++) {
      System.out.print(string2[i]);
    }
    System.out.print(", ");

    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


```
/**
 * 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' };
    char[] stringConcat = new char[string1.length + string2.length];

    concat(string1, string2, stringConcat);

    for (int i = 0; i < string1.length; i++) {
      System.out.print(string1[i]);
    }
    System.out.print(", ");

    for (int i = 0; i < string2.length; i++) {
      System.out.print(string2[i]);
    }
    System.out.print(", ");

    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];
    }
  }
}
```

# Assembly Screenshot



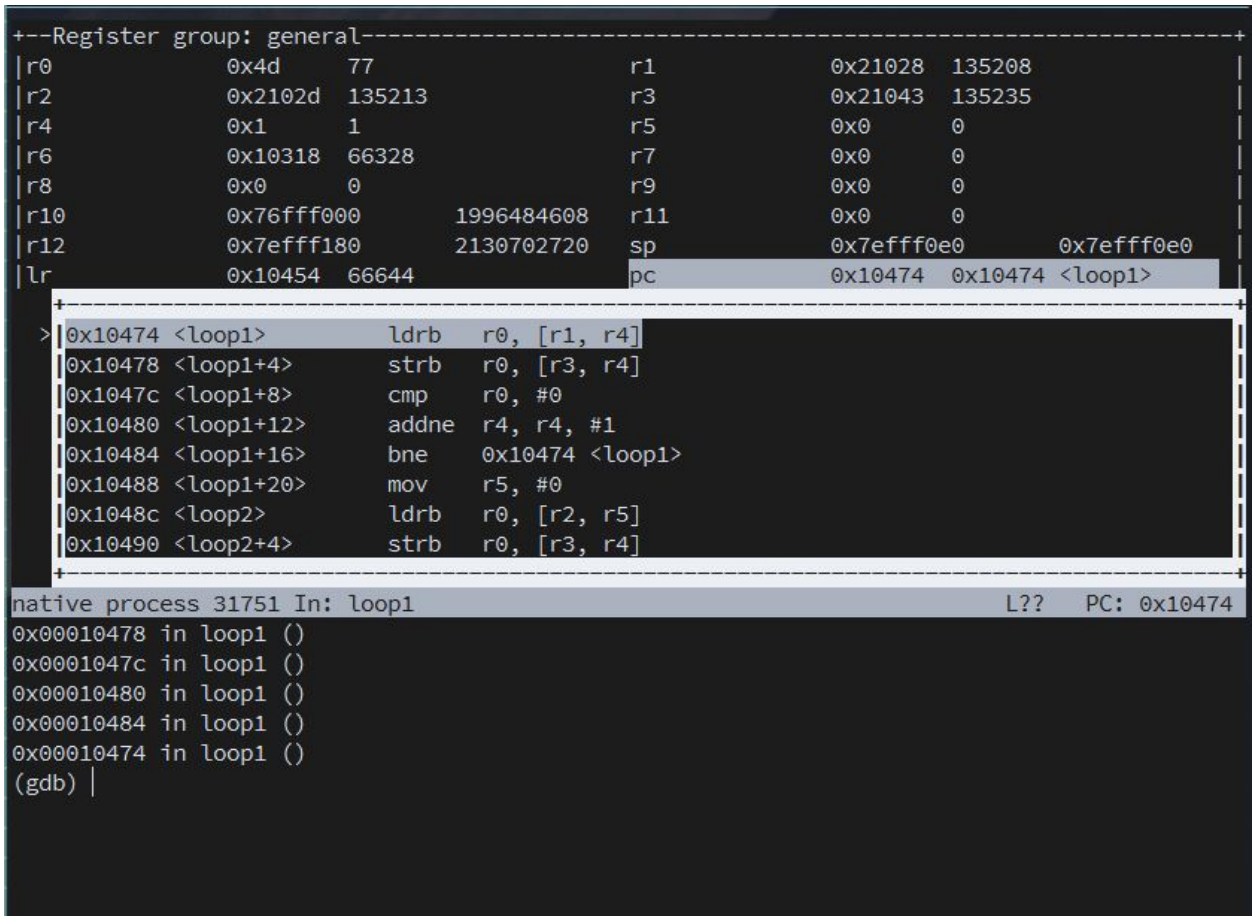
```
@ ------------------------------------------------------------------------
@ Mark Lucernas
@ Unit 8 Assignment
@ ------------------------------------------------------------------------

@ Constants
@ ------------------------------------------------------------------------
.equ EXIT,      1               @ Exit code for system call

@ Data Section
@ ------------------------------------------------------------------------
.data

@ Initialize Data
@ --------------------------------------
.balign 4
string1:        .asciz  "Mark"
string2:        .asciz  "Lucernas"
message:        .asciz  "%s, %s, %s\n"

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

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

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()

        ldr r0, =message                @ Load message (output format)
        bl printf

exit:   pop {ip, lr}                    @ Pop the values from stack into ip and pc register
        mov r0, #0                      @ r0 <- 0 (Exit code)
        mov r7, #EXIT                   @ Request to exit program
        svc 0                           @ Perform the system call

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

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

concat: push {r1, r2, r3, r4, r5, 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 {r1, r2, r3, r4, r5, lr}            @ Restore registers
        bx lr                           @ Return function

addrString1:            .int string1            @ Get address of string1
addrString2:            .int string2            @ Get address of string2
addrFinalString:        .int finalString        @ Get address of finalString
                                                                        15,1          Top
```

## Assembly Source Code

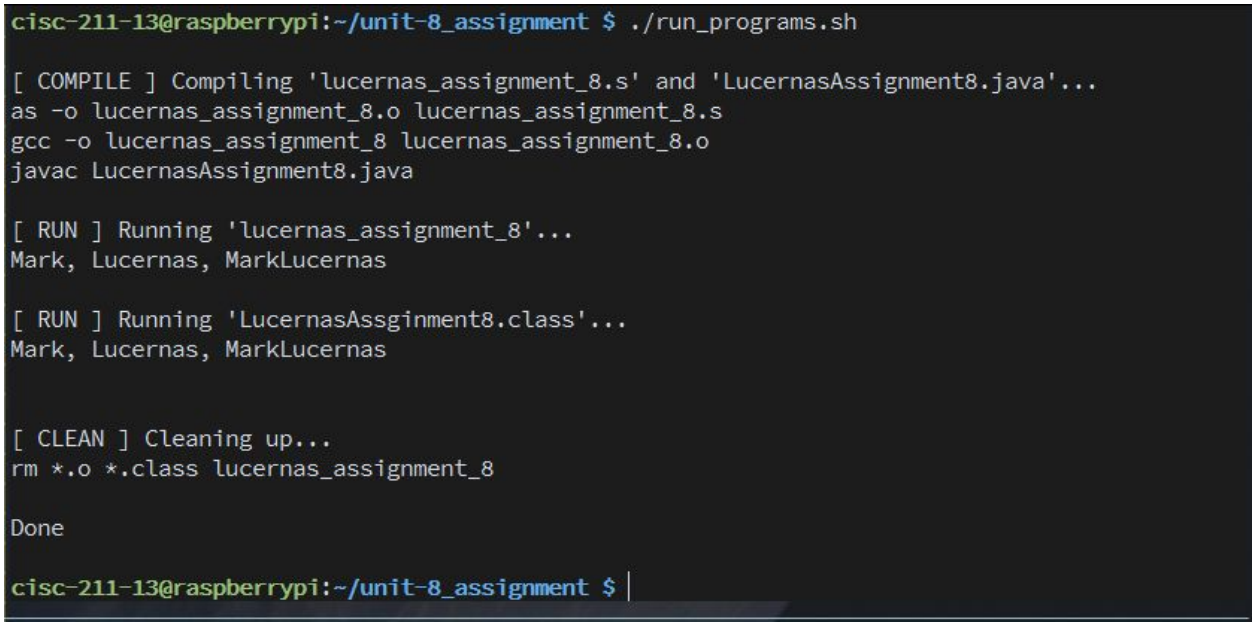
```
@ ------------------------------------------------------------------------------
@ Mark Lucernas
@ Unit 8 Assignment
@ ------------------------------------------------------------------------------

@ Constants
@ ------------------------------------------------------------------------------
.equ EXIT, 1                            @ Exit code for system call

@ Data Section
@ ------------------------------------------------------------------------------
.data

@ Initialize Data
@ --------------------------------------
.balign    4
string1:    .asciz    "Mark"
string2:    .asciz    "Lucernas"
message:    .asciz    "%s, %s, %s\n"

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

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

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

   ldr r1, addrString1        @ r1 <- string1 address
   ldr r2, addrString2        @ r2 <- string2 address
   ldr r3, addrResult         @ r3 <- result address
   bl concat                  @ Call concat()

   ldr r0, =message           @ Load message (output format)
   bl printf                  @ Print message

exit:    pop {ip, lr}                   @ Pop the values from stack into ip and pc register
   mov r0, #0                 @ r0 <- 0 (Exit code)
   mov r7, #EXIT                  @ Request to exit program
   svc 0                          @ Perform the system call
```

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

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

concat: push {r1, r2, r3, r4, r5, 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 {r1, r2, r3, r4, r5, lr}   @ Restore registers
    bx lr                       @ Return function

addrString1:    .int string1            @ Get address of string1
addrString2:    .int string2            @ Get address of string2
addrResult:    .int result      @ Get address of result
```

## GDB Debugger of Assembly Code




## Compile and Run



# Problem 2

Exercise 7.2

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