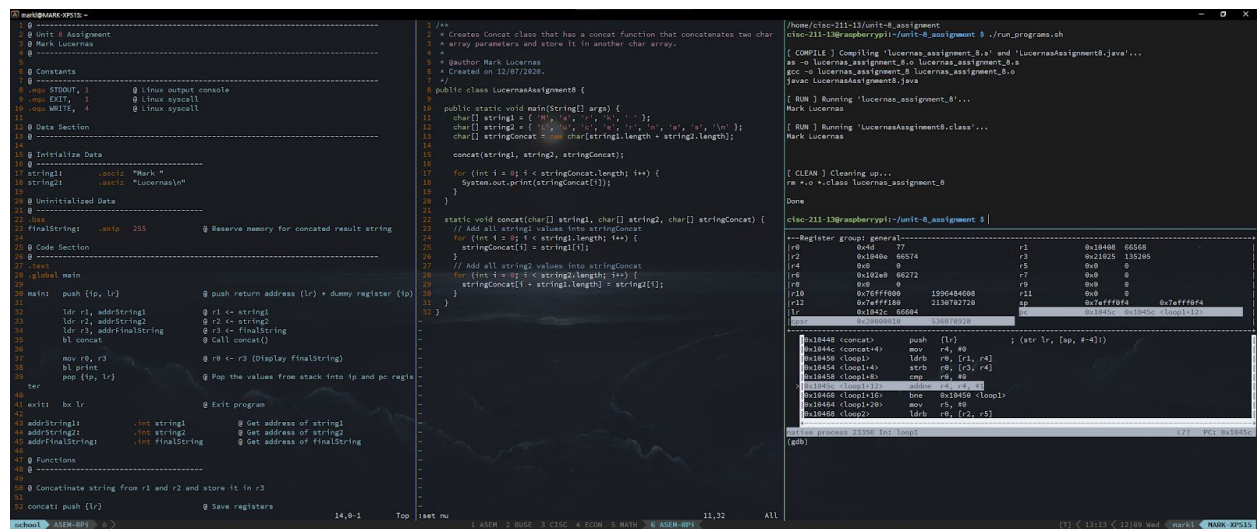


cisc-211-13@raspberrypi



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

```
1 | @ -----
2 | @ Unit 8 Assignment
3 | @ Mark Lucernas
4 | @ -----
5 |
6 | @ Constants
7 | @ -----
8 | .equ STDOUT, 1      @ Linux output console
9 | .equ EXIT, 1        @ Linux syscall
10 | .equ WRITE, 4       @ Linux syscall
11 |
12 | @ Data Section
13 | @ -----
14 |
15 | @ Initialize Data
16 | @ -----
17 | string1: .asciz "Mark "
18 | string2: .asciz "Lucernas\n"
19 |
20 | @ Uninitialized Data
21 | @ -----
22 | .bss
23 | finalString: .skip 255 @ Reserve memory for concated result string
24 |
25 | @ Code Section
26 | @ -----
27 | .text
28 | .global main
29 |
30 | main: push {ip, lr} @ push return address (lr) + dummy register (ip)
31 |
32 |     ldr r1, addrString1 @ r1 <- string1
33 |     ldr r2, addrString2 @ r2 <- string2
34 |     ldr r3, addrFinalString @ r3 <- finalString
35 |     bl concat @ Call concat()
36 |
37 |     mov r0, r3 @ r0 <- r3 (Display finalString)
38 |     bl print
39 |     pop {ip, lr} @ Pop the values from stack into ip and pc register
40 |
41 | exit: bx lr @ Exit program
42 |
43 | addrString1: .int string1 @ Get address of string1
44 | addrString2: .int string2 @ Get address of string2
45 | addrFinalString: .int finalString @ Get address of finalString
46 |
47 | @ Functions
48 | @ -----
49 |
50 | @ Concatenate string from r1 and r2 and store it in r3
51 |
52 | concat: push {lr} @ Save registers
53 |     mov r4, #0 @ r4 <- 0 (i = 0)
54 | loop1:
55 |     ldrb r0, [r1, r4] @ Load next byte (char) of string1 into r0
56 |     strb r0, [r3, r4] @ Store next byte (char) of string1 into finalString
57 |     cmp r0, #0 @ Compare if r0 is empty (no more to load from string1)
58 |     addne r4, #1 @ If not empty: r4 <- r4 + 1
59 |     bne loop1 @ Repeat loop if r0 not empty
60 |
61 |     mov r5, #0 @ r5 <- 0 (j = 0)
62 |
63 | loop2: ldrb r0, [r2, r5] @ Load next byte (char) of string2 into r0
64 |     strb r0, [r3, r4] @ Store next byte (char) of string2 into finalString
65 |     cmp r0, #0 @ Compare if r0 is empty (no more to load from string1)
66 |     addne r4, #1 @ If not empty: r4 <- r4 + 1
67 |     addne r5, #1 @ If not empty: Go to next string1 byte address
68 |     bne loop2 @ Repeat loop if r0 not empty
69 |     pop {lr} @ Restore registers
70 |     bx lr @ Return function
71 |
72 | @ Display text with size calculation
73 |
74 | print: push {r0, r1, r2, r7, lr} @ Save registers
75 |     mov r2, #0 @ r2 <- 0 (length counter)
76 | loop3: ldrb r1, [r0, r2] @ Load next byte (char) of r0 (finalString) into r1
77 |     cmp r1, #0 @ Compare if r0 is empty (no more to load from finalString)
78 |     addne r2, r2, #1 @ If not empty: r2 <- r2 + 1
79 |     bne loop3 @ Repeat loop if r0 not empty
80 |
81 |     mov r1, r0 @ r1 <- r0 (&finalString)
82 |     mov r0, #STDOUT @ Code to write to the standard output Linux
83 |     mov r7, #WRITE @ Code to system call "write"
84 |     svc #0 @ Call system
85 |     pop {r0, r1, r2, r7, lr} @ Restore registers
86 |     bx lr @ Return function
```

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

@ -----

@ Concatenate 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 (char) 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

```
+--Register group: general-----
|r0      0x6b      107                r1      0x10408  66568
|r2      0x1040e  66574                r3      0x21025  135205
|r4      0x3       3                   r5      0x0       0
|r6      0x102e0  66272                r7      0x0       0
|r8      0x0       0                   r9      0x0       0
|r10     0x76fff000 1996484608          r11     0x0       0
|r12     0x7efff180 2130702720         sp      0x7efff0f4  0x7efff0f4
|lr      0x1042c  66604                pc      0x10458  0x10458 <loop1+8>
|cpsr    0x20000010 536870928
|
-----
|0x10448 <concat>      push    {lr}          ; (str lr, [sp, #-4]!)
|0x1044c <concat+4>    mov     r4, #0
|0x10450 <loop1>       ldrb    r0, [r1, r4]
|0x10454 <loop1+4>     strb    r0, [r3, r4]
>|0x10458 <loop1+8>    cmp     r0, #0
|0x1045c <loop1+12>    addne   r4, r4, #1
|0x10460 <loop1+16>    bne     0x10450 <loop1>
|0x10464 <loop1+20>    mov     r5, #0
|0x10468 <loop2>       ldrb    r0, [r2, r5]
|0x1046c <loop2+4>     strb    r0, [r3, r4]
0x0001045c in loop1 ()
0x00010460 in loop1 ()
0x00010450 in loop1 ()
0x00010454 in loop1 ()
0x00010458 in loop1 ()
0x0001045c in loop1 ()
0x00010460 in loop1 ()
0x00010450 in loop1 ()
0x00010454 in loop1 ()
0x00010458 in loop1 ()
(gdb) |
```

High-level Language: Java Screenshot

```
1 /**
2  * Creates Concat class that has a concat function that concatenates two char
3  * array parameters and store it in another char array.
4  *
5  * @author Mark Lucernas
6  * Created on 12/07/2020.
7  */
8 public class LucernasAssignment8 {
9
10     public static void main(String[] args) {
11         char[] string1 = { 'M', 'a', 'r', 'k', ' ' };
12         char[] string2 = { 'L', 'u', 'c', 'e', 'r', 'n', 'a', 's', '\n' };
13         char[] stringConcat = new char[string1.length + string2.length];
14
15         concat(string1, string2, stringConcat);
16
17         for (int i = 0; i < stringConcat.length; i++) {
18             System.out.print(stringConcat[i]);
19         }
20     }
21
22     static void concat(char[] string1, char[] string2, char[] stringConcat) {
23         // Add all string1 values into stringConcat
24         for (int i = 0; i < string1.length; i++) {
25             stringConcat[i] = string1[i];
26         }
27         // Add all string2 values into stringConcat
28         for (int i = 0; i < string2.length; i++) {
29             stringConcat[i + string1.length] = string2[i];
30         }
31     }
32 }
```

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', '\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.