**Program 1**

Source Code

.syntax unified

**.section** **.data**

**.global** main

**.align** 4

**STATUS:** **.byte** 0x97

**.align** 4

**CONTROL:** **.byte** 0xEC

**.align** 4

**PERIPH:** **.byte** 0x15

**.text**

**main:**

// Initialize variables

ldr r0, =STATUS

ldr r1, =CONTROL

ldr r2, =PERIPH

// Load values from memory

ldrb r3, [r0]

// If bit 3 of STATUS is 1, set bit 4 of CONTROL to 1, otherwise set bit 5 of CONTROL to 0.

tst r3, #8

it eq

movseq r4, #0x20

it ne

movne r4, #0x10

ldrb r5, [r1]

orr r5, r4

strb r5, [r1]

// If bit 5 of STATUS is 0 and bit 6 of STATUS is 0, set bit 2 of CONTROL to 0, otherwise set bit 0 of CONTROL to 1.

tst r3, #0x20

bne op2\_bit6

tst r3, #0x40

it eq

beq op2\_bit6

tst r3, #0x40

it eq

movseq r4, #0xFC

it ne

movne r4, #0x01

orr r5, r4

strb r5, [r1]

**op2\_bit6:**

// Replace bits 5-2 of PERIPH with the hex digit A, without changing the other four bits of PERIPH.

mov r4, #0xA0

ldrb r6, [r2]

and r6, #0x03

orr r6, r4

strb r6, [r2]

// Effectively halts the program.

**Here:** b Here

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Program 2**

Source Code

.syntax unified

.equ NULL, 0x00

.equ MAX\_LENGTH, 20

.equ ASCII\_UPPERCASE\_A, 'A'

.equ ASCII\_UPPERCASE\_Z, 'Z'

.equ ASCII\_LOWERCASE\_A, 'a'

.equ ASCII\_LOWERCASE\_Z, 'z'

.global main

main:

// Initialize registers

ldr r0, =0x02000000 // Address of the string

mov r1, #0 // Index of the current character

convert\_loop:

// Load the current character

ldrb r2, [r0, r1]

// Check if the character is null

cmp r2, #NULL

beq exit

// Check if the character is lowercase

cmp r2, #ASCII\_LOWERCASE\_A

blt increment\_index

cmp r2, #ASCII\_LOWERCASE\_Z

bgt increment\_index

// Convert the character to uppercase

sub r2, r2, #'a'-'A'

strb r2, [r0, r1]

increment\_index:

// Increment the index

add r1, r1, #1

// Check if the maximum length has been reached

cmp r1, #MAX\_LENGTH

bge exit

// Continue to the next character

b convert\_loop

// Effectively halts the program.

**Here:** b Here

Run number 1 with this set: Includes the null char and 19 other char

set {char[20]}0x02000000 = "This is ninTeEnCHar"

Run number 2 with this set: Includes the null char and 12 other char

set {char[13]}0x02000000 = "ThirTEenChar"

**Program 3**

Source Code

.syntax unified

**.data**

**PACK:** **.word** 0x13587609

**STRG:** **.byte** 8

**.align** 4

**.text**

**.global** main

**main:**

LDR r0, =PACK

LDR r1, =STRG

MOV r2, #0

**loop:**

LDR r3, [r0]

MOV r4, #28

**digit\_loop:**

MOV r5, r3, LSR r4

AND r5, r5, #0xF

LDR r6, #0x30

ADD r5, r5, r6

STRB r5, [r1], #1

SUB r4, r4, #4

ADD r2, r2, #1

CMP r2, #8

BNE digit\_loop

MOV r5, #0

STRB r5, [r1], #1

**Here:** b Here

**Program 4**

Source Program

.syntax unified

// Main program

**.section** **.data**

**LIST1:** **.word** 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

**NPOS1:** **.word** 0

**LIST2:** **.word** 0,0,0,0,0,0,0,0,0,0,0,0

**NPOS2:** **.word** 0

**.section** **.text**

**.align**

**.global** main

**main:**

// Call the subroutine for LIST1 and store the result in NPOS1

ldr r0, =LIST1

ldr r1, =20

bl count\_nonzero\_positives

ldr r1, =NPOS1

str r0, [r1]

// Call the subroutine for LIST2 and store the result in NPOS2

ldr r0, =LIST2

ldr r1, =12

bl count\_nonzero\_positives

ldr r1, =NPOS2

str r0, [r1]

// End the program

mov r0, #0

bx lr

// Subroutine to count the number of non-zero positive numbers in an array of signed 32-bit integers

**count\_nonzero\_positives:**

mov r2, #0

**loop:**

ldr r3, [r0], #4 // Load the next element of array. increment the pointer next element

cmp r3, #0 // Compare the loaded value with zero

blt skip // If it's negative, skip to the next iteration

add r2, r2, #1

**skip:**

subs r1, r1, #1 // Decrement the size counter

bne loop // If there are still elements left, repeat the loop

mov r0, r2

bx lr

Graphical user interface, text, application

Description automatically generated