INTERNATIONAL UNIVERSITY OF SARAJEVO

MIPS Assembly Project - Palindrome

CS304 Computer architecture

Kenan Klepić, 200302040

Sarajevo, Spring 2022

Contents

1. Table of Figures

2. Project Description

The name of this project, for the CS304 Computer Architecture, also identifies the purpose and functionality of this project. Namely, this project titled Palindrome, has the goal of creating a program in MIPS, that is in the assembly language, which is capable of determining whether a certain string can be considered a palindrome. The idea of a palindrome project is not particularly innovative or difficult to accomplish, and most beginner programmers create this type of program in a programming language that they are trying to learn. However, the challenge of this project lays in the fact that the task is not to create a palindrome checking program in a high-level language such as C, Java or Python, rather the task is to create a fully functioning palindrome program in assembly language. Furthermore, because the task is to use assembly language, the goal of this project is to see how this program functions under the hood, that is how does the processor interpret complex instructions, how do the registers behave and what are they used for.

Before introducing the code, it is necessary to know what is a palindrome and what does a palindrome program do. A palindrome represents a string of characters, alphabetical or numerical, that is the same when reads forwards and backwards. This means that if we were to have a word that is a palindrome, even if we read it from the last character to the first character, the word would be the same as the original word. Some examples of palindromes include words such as: noon, radar, deed, as well as numbers such as: 101, 4554, 77 etc. In this project, the task was to focus on alphabetical words, and to have these words as inputs. Then, the program should determine whether the inputted word is a palindrome or not.

For example, if the input is “noon” the program should display that this word is a palindrome (with words or numbers, or other values), and if he input is “moon” the program should in some way indicate that this word is not a palindrome.

3. Program Design

To better understand how the functionality of this program is implemented it is important to create a flowchart for the code algorithm. Since the algorithm for calculating the string length was covered by the professor Khaldoun Al Khalidi, and the code for this part was provided for the project, this report will not explain this part of the code, rather the code that determines whether the string is a palindrome. Note, the code for string length calculation, provided by professor Khaldoun Al Khalidi, was used in this project.

The algorithm for determining whether the string is a palindrome, with provided string length, can be seen below.

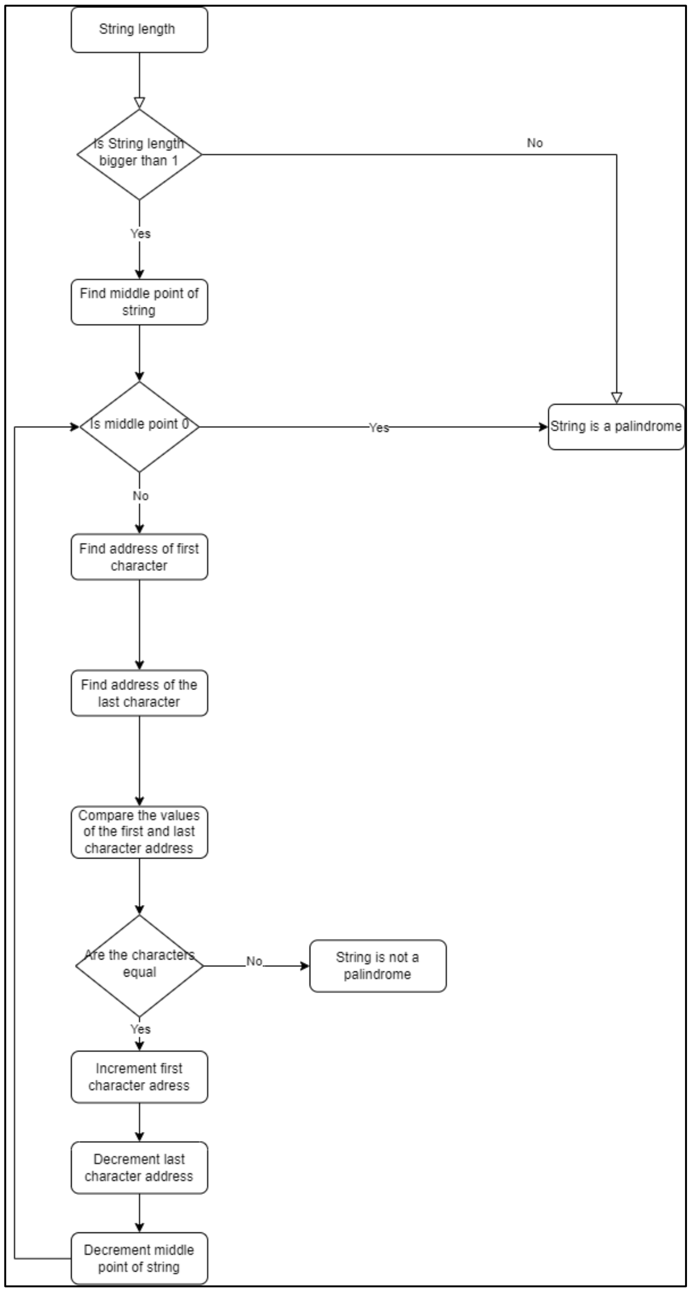

String length
Is String length bigger than 1
No
Yes
Find middle point of string
Is middle point 0
Yes
String is a palindrome
No
Find address of first character
Find address of the last character
Compare the values of the first and last character address
Are the characters equal
No
String is not a palindrome
Yes
Increment first character adress
Decrement last character address
Decrement middle point of string

Figure 1. Palindrome Algorithm

From this algorithm one can observe the logic of this code. The first step is to check whether the string length is 0 or 1, if it is the program doesn't do any checking because a string has to have at least 2 characters to be in consideration for a word that is not a palindrome. If the string length is 0 or 1 the code treats this word as a palindrome. If, however, the string has more than 1 character, the code will do some checking. Firstly, it is important to determine the middle point of the string by dividing the string length with 2. The reason for this middle point will be explained in the next steps. Then, the code stores the address that points to the first and the last character in the string. This step is necessary since by doing this the code can compare the values of these addresses to determine whether they have the same value. The code in this project tracks the first character address by pointing to the first address in the data section of the processor, since this where the input string will be kept. The last character address is assigned when the string length is calculated. Since in the length calculation process, the first character address pointer is incremented until it reaches the null terminator “\n”, and the since the null terminator indicated the end of the string, at the end of the length calculation this incremented address will point to the last character address. As previously mentioned, the code will then compare the values of the first and last character address, and if they are not the same, this indicates that the word is not a palindrome and the program can end. If the values are the same, the code then needs to check the second and second to last character for equality. The code will do this with a loop, but before the loop, the first character address will be incremented so that it now points to the second character address, and the last character address will be decremented so that it points to the second to last character address. The middle point of the string will also be decremented in the next step, because this number, the middle point, is used to determine how many characters do we have to compare for equality. This stems from the fact that if the middle point is zero, this means that the two inner most characters of the string were compared, and that there are no more characters between these two, in case that the string length is even. If the string length number is odd, then there will be one more character between the two inner most characters, but since this middle character will be at the same position when the word is read forwards and backwards, there is no need to check this character. Because of this, the code uses the middle point to determine how many times the loop will be iterated, that is, if the middle point reaches zero then this will indicate that the outer most character pairs are equal, and in turn this means that the word is a palindrome.

4. Symbol Table

Many registers were used for this code to function properly, and all of these registers, their address and purpose, are displayed in the table below.

Table 1. Register Symbol Table.

Register	Description
$0	zero register, holds all zeros, mainly used for comparisons
$1	register reserved for the assembler
$2	register used for system calls
$4	output register, holds the result of the program; if the output is 1 word is a palindrome, if output is 0 word is not a palindrome
$8	used to store string length and to store middle point of the string
$9	pointer to the first character; in the palindrome part of the code pointer to the last character
$10	stores the value (the character) of the pointer register
$11	pointer to the first character
$13	holds value 0 or 1, 1 if the string length is smaller than 2, 0 if the length is 2 or bigger, used for checking string length
$12	stores the value (the character) of the pointer register

Besides the registers, this code also uses labels primarily used for jumping to a certain part of the code. The labels are displayed in the table below.

Table 2. Symbol table for Labels

Label	Description

main	initialization and setting the pointers
loop	checking the string length
done	indicates that the string length is calculated
palindrome	check string length, determine the middle point
palLoop	comparing tevery farthest right and farthest left character, used for determining if the word is a palindrome
finishTrue	adds 1 to the output register, indicates that the word passed the palindrome test
finishFalse	adds 0 to the output register, indicates that the word failed the palindrome test
string	used for string input

5. Prototype

The prototype for this project was done in C programming language, since this language resembles the assembly language in its syntax. This prototype was used to determine the algorithm logic; therefore, the C code uses the same logic already explained in the Project Design part of the report. The C code can be seen below.

```
#include <stdio.h>
#include <stdlib.h>


int main()
{


    char s[] = "geteg";
    int palindrome = 1;
    int length = 0;


    printf("%s", s);
    while (s[length] != '\0')
    {
        length++;
    }

    int halfLen = length/2;

    int begin = 0;
    int end = length-1;

    while(halfLen != 0)
    {
        if(s[begin] != s[end])
        {
            palindrome = 0;
            break;
        }

        begin++;
        end--;
        halfLen--;
    }

    if (palindrome == 1)
        printf("\nPalindrome");
    else printf("\nNot Palindrome");

    return 0;
}
```

6. Test Plan

The string chosen to be the test string for this project are: “deed”, “dead”, “d” and “ ”.

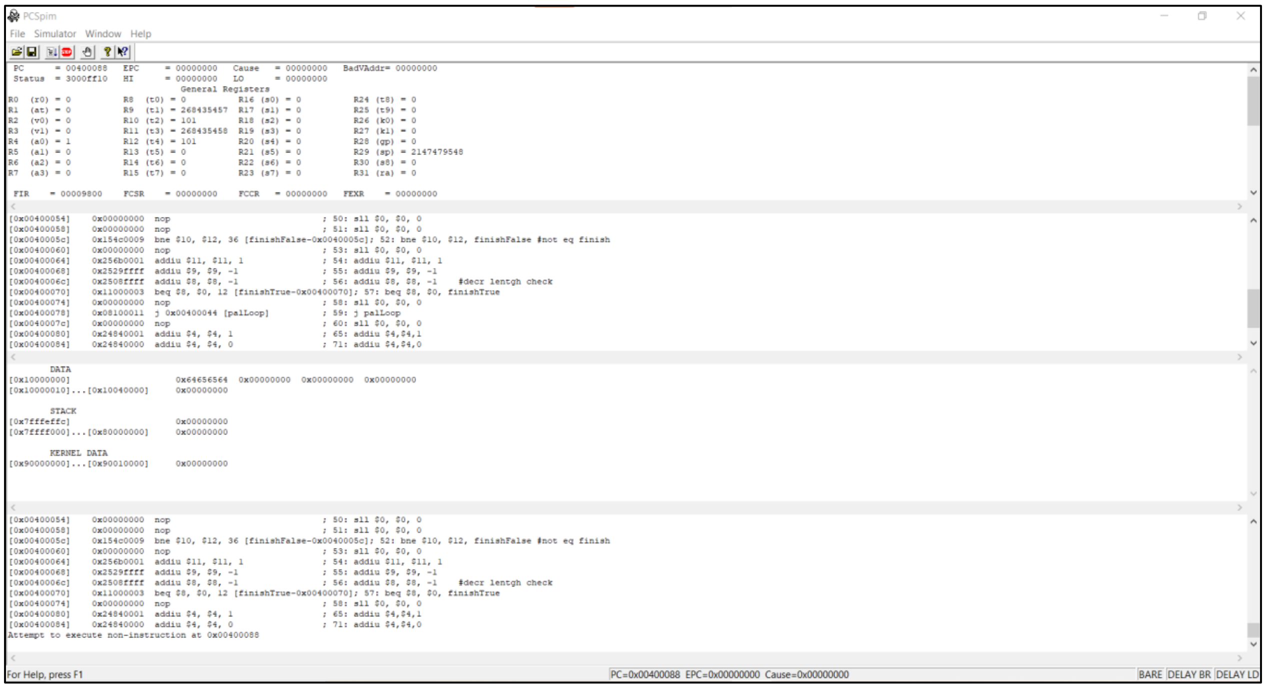
The reasoning for using the word “deed” is somewhat clear, this word is a palindrome and it can be used to check whether the program recognizes palindromes. The value stored in the $4 register should be 1 for this string.

Next test string is “dead”. This word is not a palindrome, and it checks whether the code recognizes words that are not palindromes. The expected output of this word is the value 0 in the $4 register.

Lastly, a string with one character “d” is used to see whether the part of the code that checks the string length works. The output of this string should be 1 in $4. The blank string “ “, with no value, is used for the same reason as the string with only one character and the output of these strings should be the same – 1 in $4.

7. Test Results

In this part of the report, string previously chosen in the Test Plan part, will be used inputted into the project code, and the code will be simulated in the MIPS simulator – SPIM to see whether the code will produce the expected results. The code used and created for this project can be found in the Appendix A of this report.



```
 PC      = 00400088   EPC     = 00000000   Cause    = 00000000   BadVAddr= 00000000
 Status  = 3000ff10   HI      = 00000000   LO       = 00000000
                           General Registers
R0  (r0) = 0          R8  (t0) = 0          R16 (s0) = 0          R24 (t8) = 0
R1  (at) = 0          R9  (t1) = 268435457  R17 (s1) = 0          R25 (t9) = 0
R2  (v0) = 0          R10 (t2) = 101        R18 (s2) = 0          R26 (k0) = 0
R3  (v1) = 0          R11 (t3) = 268435458  R19 (s3) = 0          R27 (k1) = 0
R4  (a0) = 1          R12 (t4) = 101        R20 (s4) = 0          R28 (gp) = 0
R5  (a1) = 0          R13 (t5) = 0          R21 (s5) = 0          R29 (sp) = 2147479548
R6  (a2) = 0          R14 (t6) = 0          R22 (s6) = 0          R30 (s8) = 0
R7  (a3) = 0          R15 (t7) = 0          R23 (s7) = 0          R31 (ra) = 0

 FIR     = 00009800   FCSR    = 00000000   FCCR     = 00000000   FEXR     = 00000000
```

```
[0x00400054]    0x00000000  nop                                     ; 50: sll $0, $0, 0
[0x00400058]    0x00000000  nop                                     ; 51: sll $0, $0, 0
[0x0040005c]    0x154c0009  bne $10, $12, 36 [finishFalse-0x0040005c]; 52: bne $10, $12, finishFalse #not eq finish
[0x00400060]    0x00000000  nop                                     ; 53: sll $0, $0, 0
[0x00400064]    0x256b0001  addiu $11, $11, 1                       ; 54: addiu $11, $11, 1
[0x00400068]    0x2529ffff  addiu $9, $9, -1                        ; 55: addiu $9, $9, -1
[0x0040006c]    0x2508ffff  addiu $8, $8, -1                        ; 56: addiu $8, $8, -1    #decr lentgh check
[0x00400070]    0x11000003  beq $8, $0, 12 [finishTrue-0x00400070]; 57: beq $8, $0, finishTrue
[0x00400074]    0x00000000  nop                                     ; 58: sll $0, $0, 0
[0x00400078]    0x08100011  j 0x00400044 [palLoop]                  ; 59: j palLoop
[0x0040007c]    0x00000000  nop                                     ; 60: sll $0, $0, 0
[0x00400080]    0x24840001  addiu $4, $4, 1                         ; 65: addiu $4,$4,1
[0x00400084]    0x24840000  addiu $4, $4, 0                         ; 71: addiu $4,$4,0
```

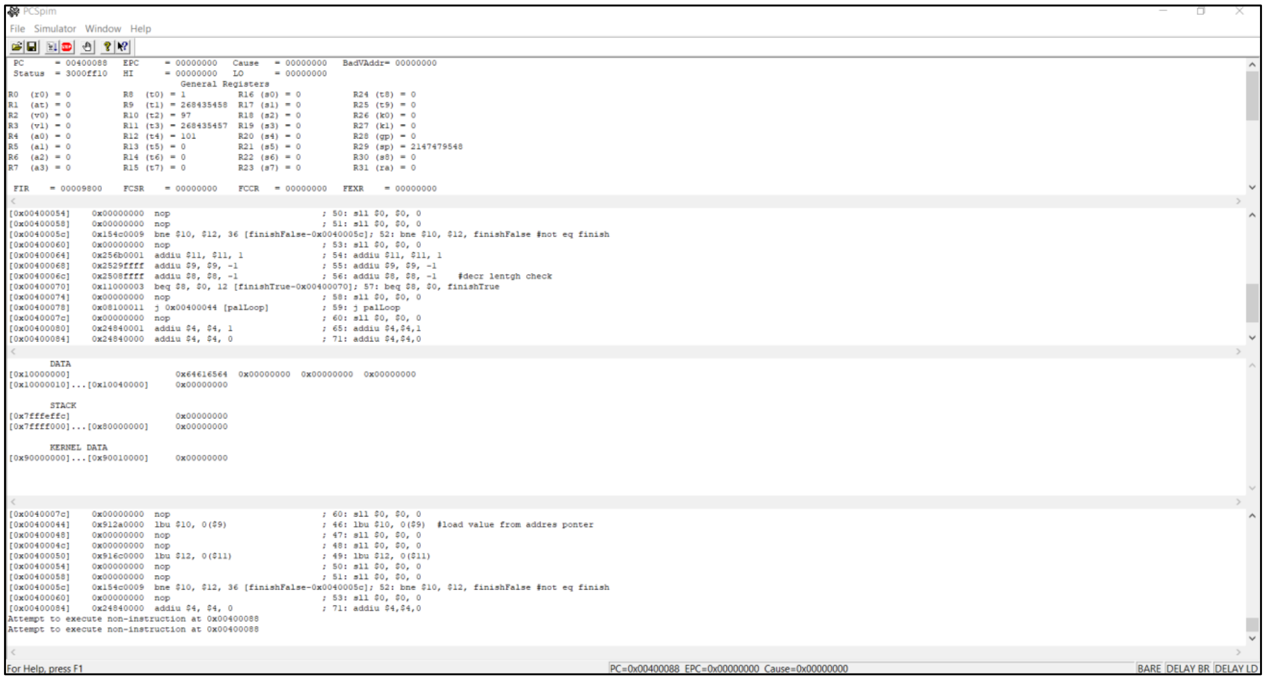
```
        DATA
[0x10000000]                    0x64656564  0x00000000  0x00000000  0x00000000
[0x10000010]...[0x10040000]     0x00000000

        STACK
[0x7fffeffc]                    0x00000000
[0x7ffff000]...[0x80000000]     0x00000000

        KERNEL DATA
[0x90000000]...[0x90010000]     0x00000000
```

```
[0x00400054]    0x00000000  nop                                     ; 50: sll $0, $0, 0
[0x00400058]    0x00000000  nop                                     ; 51: sll $0, $0, 0
[0x0040005c]    0x154c0009  bne $10, $12, 36 [finishFalse-0x0040005c]; 52: bne $10, $12, finishFalse #not eq finish
[0x00400060]    0x00000000  nop                                     ; 53: sll $0, $0, 0
[0x00400064]    0x256b0001  addiu $11, $11, 1                       ; 54: addiu $11, $11, 1
[0x00400068]    0x2529ffff  addiu $9, $9, -1                        ; 55: addiu $9, $9, -1
[0x0040006c]    0x2508ffff  addiu $8, $8, -1                        ; 56: addiu $8, $8, -1    #decr lentgh check
[0x00400070]    0x11000003  beq $8, $0, 12 [finishTrue-0x00400070]; 57: beq $8, $0, finishTrue
[0x00400074]    0x00000000  nop                                     ; 58: sll $0, $0, 0
[0x00400080]    0x24840001  addiu $4, $4, 1                         ; 65: addiu $4,$4,1
[0x00400084]    0x24840000  addiu $4, $4, 0                         ; 71: addiu $4,$4,0
Attempt to execute non-instruction at 0x00400088
```

```
For Help, press F1          PC=0x00400088  EPC=0x00000000  Cause=0x00000000          BARE  DELAY BR  DELAY LD
```

Figure 2. First Test Result

For the first test, the used string was “deed”, and the expected result for this word was 1, that is the indicator that the word is a palindrome. It can be seen that the register $4 has the value 1, meaning that the code passed this test.

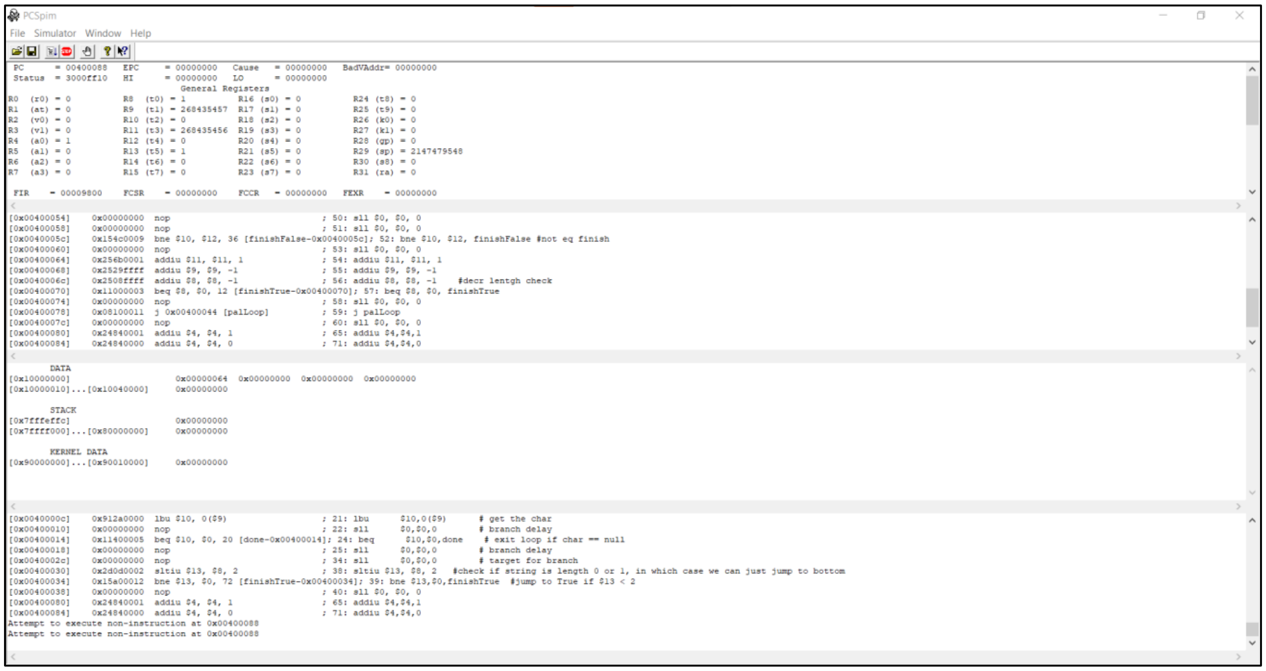


```
PC       = 00400088   EPC      = 00000000   Cause    = 00000000   BadVAddr= 00000000
Status   = 3000ff10   HI       = 00000000   LO       = 00000000
                               General Registers
R0  (r0) = 0          R8  (t0) = 1          R16 (s0) = 0          R24 (t8) = 0
R1  (at) = 0          R9  (t1) = 268435458  R17 (s1) = 0          R25 (t9) = 0
R2  (v0) = 0          R10 (t2) = 97         R18 (s2) = 0          R26 (k0) = 0
R3  (v1) = 0          R11 (t3) = 268435457  R19 (s3) = 0          R27 (k1) = 0
R4  (a0) = 0          R12 (t4) = 101        R20 (s4) = 0          R28 (gp) = 0
R5  (a1) = 0          R13 (t5) = 0          R21 (s5) = 0          R29 (sp) = 2147479548
R6  (a2) = 0          R14 (t6) = 0          R22 (s6) = 0          R30 (s8) = 0
R7  (a3) = 0          R15 (t7) = 0          R23 (s7) = 0          R31 (ra) = 0

FIR    = 00009800     FCSR     = 00000000   FCCR     = 00000000   FEXR     = 00000000

[0x00400054]    0x00000000  nop                              ; 50: sll $0, $0, 0
[0x00400058]    0x00000000  nop                              ; 51: sll $0, $0, 0
[0x0040005c]    0x154c0009  bne $10, $12, 36 [finishFalse-0x0040005c]; 52: bne $10, $12, finishFalse #not eq finish
[0x00400060]    0x00000000  nop                              ; 53: sll $0, $0, 0
[0x00400064]    0x256b0001  addiu $11, $11, 1                ; 54: addiu $11, $11, 1
[0x00400068]    0x2529ffff  addiu $9, $9, -1                 ; 55: addiu $9, $9, -1
[0x0040006c]    0x2508ffff  addiu $8, $8, -1                 ; 56: addiu $8, $8, -1    #decr lentgh check
[0x00400070]    0x11000003  beq $8, $0, 12 [finishTrue-0x00400070]; 57: beq $8, $0, finishTrue
[0x00400074]    0x00000000  nop                              ; 58: sll $0, $0, 0
[0x00400078]    0x08100011  j 0x00400044 [palLoop]           ; 59: j palLoop
[0x0040007c]    0x00000000  nop                              ; 60: sll $0, $0, 0
[0x00400080]    0x24840001  addiu $4, $4, 1                  ; 65: addiu $4,$4,1
[0x00400084]    0x24840000  addiu $4, $4, 0                  ; 71: addiu $4,$4,0

        DATA
[0x10000000]                    0x64616564  0x00000000  0x00000000  0x00000000
[0x10000010]...[0x10040000]     0x00000000

        STACK
[0x7fffeffc]                    0x00000000
[0x7ffff000]...[0x80000000]     0x00000000

        KERNEL DATA
[0x90000000]...[0x90010000]     0x00000000

[0x0040007c]    0x00000000  nop                              ; 60: sll $0, $0, 0
[0x00400044]    0x912a0000  lbu $10, 0($9)                   ; 46: lbu $10, 0($9)  #load value from addres ponter
[0x00400048]    0x00000000  nop                              ; 47: sll $0, $0, 0
[0x0040004c]    0x00000000  nop                              ; 48: sll $0, $0, 0
[0x00400050]    0x916c0000  lbu $12, 0($11)                  ; 49: lbu $12, 0($11)
[0x00400054]    0x00000000  nop                              ; 50: sll $0, $0, 0
[0x00400058]    0x00000000  nop                              ; 51: sll $0, $0, 0
[0x0040005c]    0x154c0009  bne $10, $12, 36 [finishFalse-0x0040005c]; 52: bne $10, $12, finishFalse #not eq finish
[0x00400060]    0x00000000  nop                              ; 53: sll $0, $0, 0
[0x00400084]    0x24840000  addiu $4, $4, 0                  ; 71: addiu $4,$4,0
Attempt to execute non-instruction at 0x00400088
Attempt to execute non-instruction at 0x00400088
```

For Help, press F1   PC=0x00400088 EPC=0x00000000 Cause=0x00000000   BARE DELAY BR DELAY LD

Figure 3. Second Test Result

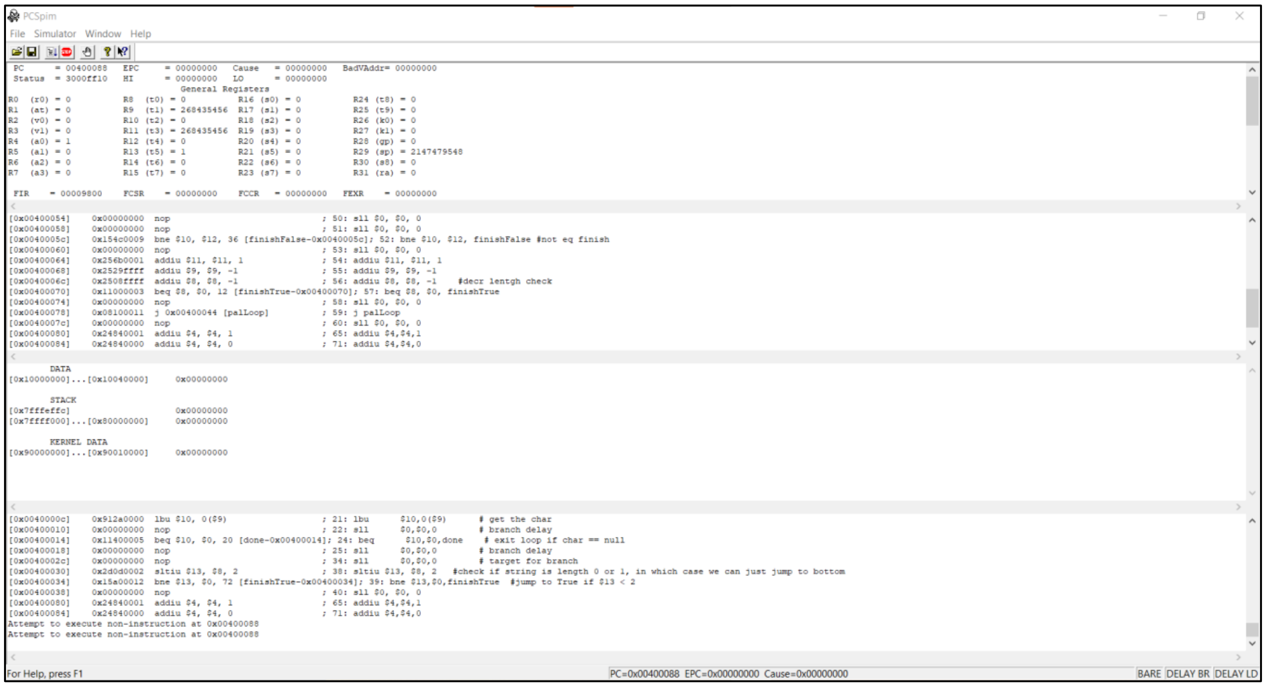
The second test word was "dead" and the expected output is 0, since this word is not a palindrome. It can be seen that the result matches the expected value, that is register $4 holds the value 0, meaning that the code passed this test.



```
PC       = 00400088   EPC      = 00000000   Cause    = 00000000   BadVAddr= 00000000
Status   = 3000ff10   HI       = 00000000   LO       = 00000000
                               General Registers
R0  (r0) = 0          R8  (t0) = 1          R16 (s0) = 0          R24 (t8) = 0
R1  (at) = 0          R9  (t1) = 268435457  R17 (s1) = 0          R25 (t9) = 0
R2  (v0) = 0          R10 (t2) = 0          R18 (s2) = 0          R26 (k0) = 0
R3  (v1) = 0          R11 (t3) = 268435456  R19 (s3) = 0          R27 (k1) = 0
R4  (a0) = 1          R12 (t4) = 0          R20 (s4) = 0          R28 (gp) = 0
R5  (a1) = 0          R13 (t5) = 1          R21 (s5) = 0          R29 (sp) = 2147479548
R6  (a2) = 0          R14 (t6) = 0          R22 (s6) = 0          R30 (s8) = 0
R7  (a3) = 0          R15 (t7) = 0          R23 (s7) = 0          R31 (ra) = 0

FIR    = 00009800     FCSR     = 00000000   FCCR     = 00000000   FEXR     = 00000000

[0x00400054]    0x00000000  nop                              ; 50: sll $0, $0, 0
[0x00400058]    0x00000000  nop                              ; 51: sll $0, $0, 0
[0x0040005c]    0x154c0009  bne $10, $12, 36 [finishFalse-0x0040005c]; 52: bne $10, $12, finishFalse #not eq finish
[0x00400060]    0x00000000  nop                              ; 53: sll $0, $0, 0
[0x00400064]    0x256b0001  addiu $11, $11, 1                ; 54: addiu $11, $11, 1
[0x00400068]    0x2529ffff  addiu $9, $9, -1                 ; 55: addiu $9, $9, -1
[0x0040006c]    0x2508ffff  addiu $8, $8, -1                 ; 56: addiu $8, $8, -1    #decr lentgh check
[0x00400070]    0x11000003  beq $8, $0, 12 [finishTrue-0x00400070]; 57: beq $8, $0, finishTrue
[0x00400074]    0x00000000  nop                              ; 58: sll $0, $0, 0
[0x00400078]    0x08100011  j 0x00400044 [palLoop]           ; 59: j palLoop
[0x0040007c]    0x00000000  nop                              ; 60: sll $0, $0, 0
[0x00400080]    0x24840001  addiu $4, $4, 1                  ; 65: addiu $4,$4,1
[0x00400084]    0x24840000  addiu $4, $4, 0                  ; 71: addiu $4,$4,0

        DATA
[0x10000000]                    0x00000064  0x00000000  0x00000000  0x00000000
[0x10000010]...[0x10040000]     0x00000000

        STACK
[0x7fffeffc]                    0x00000000
[0x7ffff000]...[0x80000000]     0x00000000

        KERNEL DATA
[0x90000000]...[0x90010000]     0x00000000

[0x0040000c]    0x912a0000  lbu $10, 0($9)                   ; 21: lbu      $10,0($9)      # get the char
[0x00400010]    0x00000000  nop                              ; 22: sll      $0,$0,0        # branch delay
[0x00400014]    0x11400005  beq $10, $0, 20 [done-0x00400014]; 24: beq      $10,$0,done    # exit loop if char == null
[0x00400018]    0x00000000  nop                              ; 25: sll      $0,$0,0        # branch delay
[0x0040002c]    0x00000000  nop                              ; 34: sll      $0,$0,0        # target for branch
[0x00400030]    0x2d0d0002  sltiu $13, $8, 2                 ; 38: sltiu $13, $8, 2   #check if string is length 0 or 1, in which case we can just jump to bottom
[0x00400034]    0x15a00012  bne $13, $0, 72 [finishTrue-0x00400034]; 39: bne $13,$0,finishTrue  #jump to True if $13 < 2
[0x00400038]    0x00000000  nop                              ; 40: sll $0, $0, 0
[0x00400080]    0x24840001  addiu $4, $4, 1                  ; 65: addiu $4,$4,1
[0x00400084]    0x24840000  addiu $4, $4, 0                  ; 71: addiu $4,$4,0
Attempt to execute non-instruction at 0x00400088
Attempt to execute non-instruction at 0x00400088
```

Figure 4. Third Test Result

The test for the one character uses the string “d”, and the expected result is 1 in the $4. It can be seen that the code passed this test since the value of register $4 is 1.



```
 PC       = 00400088   EPC      = 00000000   Cause    = 00000000   BadVAddr= 00000000
 Status   = 3000ff10   HI       = 00000000   LO       = 00000000
                                General Registers
R0  (r0) = 0           R8  (t0) = 0          R16 (s0) = 0          R24 (t8) = 0
R1  (at) = 0           R9  (t1) = 268435456  R17 (s1) = 0          R25 (t9) = 0
R2  (v0) = 0           R10 (t2) = 0          R18 (s2) = 0          R26 (k0) = 0
R3  (v1) = 0           R11 (t3) = 268435456  R19 (s3) = 0          R27 (k1) = 0
R4  (a0) = 1           R12 (t4) = 0          R20 (s4) = 0          R28 (gp) = 0
R5  (a1) = 0           R13 (t5) = 1          R21 (s5) = 0          R29 (sp) = 2147479548
R6  (a2) = 0           R14 (t6) = 0          R22 (s6) = 0          R30 (s8) = 0
R7  (a3) = 0           R15 (t7) = 0          R23 (s7) = 0          R31 (ra) = 0

 FIR    = 00009800     FCSR     = 00000000   FCCR     = 00000000   FEXR     = 00000000
```

```
[0x00400054]    0x00000000  nop                              ; 50: sll $0, $0, 0
[0x00400058]    0x00000000  nop                              ; 51: sll $0, $0, 0
[0x0040005c]    0x154c0009  bne $10, $12, 36 [finishFalse-0x0040005c]; 52: bne $10, $12, finishFalse #not eq finish
[0x00400060]    0x00000000  nop                              ; 53: sll $0, $0, 0
[0x00400064]    0x256b0001  addiu $11, $11, 1                ; 54: addiu $11, $11, 1
[0x00400068]    0x2529ffff  addiu $9, $9, -1                 ; 55: addiu $9, $9, -1
[0x0040006c]    0x2508ffff  addiu $8, $8, -1                 ; 56: addiu $8, $8, -1    #decr lentgh check
[0x00400070]    0x11000003  beq $8, $0, 12 [finishTrue-0x00400070]; 57: beq $8, $0, finishTrue
[0x00400074]    0x00000000  nop                              ; 58: sll $0, $0, 0
[0x00400078]    0x08100011  j 0x00400044 [palLoop]           ; 59: j palLoop
[0x0040007c]    0x00000000  nop                              ; 60: sll $0, $0, 0
[0x00400080]    0x24840001  addiu $4, $4, 1                  ; 65: addiu $4,$4,1
[0x00400084]    0x24840000  addiu $4, $4, 0                  ; 71: addiu $4,$4,0
```

```
        DATA
[0x10000000]...[0x10040000]     0x00000000

        STACK
[0x7fffeffc]                    0x00000000
[0x7ffff000]...[0x80000000]     0x00000000

        KERNEL DATA
[0x90000000]...[0x90010000]     0x00000000
```

```
[0x0040000c]    0x912a0000  lbu $10, 0($9)                  ; 21: lbu      $10,0($9)      # get the char
[0x00400010]    0x00000000  nop                             ; 22: sll      $0,$0,0        # branch delay
[0x00400014]    0x11400005  beq $10, $0, 20 [done-0x00400014]; 24: beq      $10,$0,done    # exit loop if char == null
[0x00400018]    0x00000000  nop                             ; 25: sll      $0,$0,0        # branch delay
[0x0040002c]    0x00000000  nop                             ; 34: sll      $0,$0,0        # target for branch
[0x00400030]    0x2d0d0002  sltiu $13, $8, 2                ; 38: sltiu $13, $8, 2   #check if string is length 0 or 1, in which case we can just jump to bottom
[0x00400034]    0x15a00012  bne $13, $0, 72 [finishTrue-0x00400034]; 39: bne $13,$0,finishTrue  #jump to True if $13 < 2
[0x00400038]    0x00000000  nop                             ; 40: sll $0, $0, 0
[0x00400080]    0x24840001  addiu $4, $4, 1                 ; 65: addiu $4,$4,1
[0x00400084]    0x24840000  addiu $4, $4, 0                 ; 71: addiu $4,$4,0
Attempt to execute non-instruction at 0x00400088
Attempt to execute non-instruction at 0x00400088
```

```
For Help, press F1          PC=0x00400088  EPC=0x00000000  Cause=0x00000000          BARE  DELAY BR  DELAY LD
```

Figure 5. Fourth Test Result

The last test was a blank string “ “ and the result in the register $4 is 1, which is the expected result determined by the program code.

Looking at the results, it can be seen that this code passed all the test meaning that its functionality is valid and that the code implementation is correct. Note, all the results were obtained by using the “Step by Step” function in the SPIM simulator.

Appendix A

```
## strlen.asm

##

## Count the characters in a string

##

## Registers:

##  $8 -- count

##  $9 -- pointer to the char

## $10 -- the char (in low order byte)


        .text

        .globl  main


# Initialize

main:


# ori     $2,$2,0

 ori     $8,$0,0       #  count = 0

        lui     $9,0x1000     #  point at first char. Change to Ox1000 in spim

        lui     $11, 0x1000   # keep track of first char address. pointer to the first char

# while not ch==null do

loop:   lbu     $10,0($9)      # get the char

        sll     $0,$0,0        # branch delay


        beq     $10,$0,done    # exit loop if char == null

        sll     $0,$0,0        # branch delay


        addiu   $8,$8,1        # count++

        addiu   $9,$9,1        # point at the next char
        j       loop
        sll     $0,$0,0        # branch delay slot

# finish
done:   sll     $0,$0,0        # target for branch

palindrome:

            sltiu $13, $8, 2   #check if string is length smaller than 2, if it is put 1 in $13, else put 0 in $13
            bne $13,$0,finishTrue  #jump to finishTrue if $13 has value 1, that is not equal to 0
            sll $0, $0, 0                    #branch delay slot
            addiu $9, $9, -1      #decrement pointer to last char because the last char is /0
            srl $8, $8, 1         #devide string with 2, find middle point

palLoop:

            lbu $10, 0($9)  #load value from last char address pointer into $10
        sll $0, $0, 0   #branch delay slot
        sll $0, $0, 0   #branch delay slot
            lbu $12, 0($11) #load value from first char address pointer into $12
        sll $0, $0, 0   #branch delay slot
        sll $0, $0, 0   #branch delay slot
            bne $10, $12, finishFalse #if first and last char are not equal then jum to finishFalse
            sll $0, $0, 0   #branch delay slot
            addiu $11, $11, 1 #increment first char address pointer
            addiu $9, $9, -1  #decrement last char address pointer
            addiu $8, $8, -1    #decrement middle point, lenght check
            beq $8, $0, finishTrue #if middle point is then jump to finshTrue
            sll $0, $0, 0   #branch delay slot
            j palLoop       #jump to begging of the loop so that the char values can be checked for equality
            sll $0, $0, 0   #branch delay slot



finishTrue:


addiu $4,$4,1   #add 1 to the output register $4




finishFalse:


addiu $4,$4,0  #add 0 to the output register $4


        .data
string:  .asciiz  "String Input"
```