

Name: Ekansh Somani

Roll Number: 20110065

Github: [Assignment 3 Code](#)

Course: ES 215: Computer Organization and Architecture

Note: Comments in the code might end up getting truncated in the report. Please refer to the github repository in case the code isn't printed correctly.

Table of Contents

- [Table of Contents](#)
- [Q1](#)
- [Q2](#)
- [Q3](#)
- [Q4](#)
- [Q5](#)
- [Q6](#)

Q1

Write a program in assembly language to subtract two 16 bit numbers without using the subtraction instruction. Note: the numbers have to be fetched from the memory.

C++ Code:

```
int subtract(int16_t a, int16_t b)
{
    int32_t p = a; // sign extension from
    int32_t q = b; // 16 to 32 bits

    q = ~q;
    q = q + 1;
    return p+q;
}
```

MIPS Assembly Code:

```
.data
PQ:      .half    10, 5           # Example value for p and q
RESULT:  .word    0              # To store the result

.text

.globl  main
main:

    la      $gp,      PQ          # load address pq into gp
    jal     SUBTRACT        # Call the Subtract function
    sw      $s0,      RESULT      # Store the result back to memory
    add     $a0,      $s0,      $zero # loading into a0 for printing

    li      $v0,      1          # print integer syscall code
    syscall

    li      $v0,      10
    syscall

SUBTRACT:

    lh      $s0,      0($gp)      # Load and sign extend a to word
    lh      $s1,      2($gp)      # Load and sign extend b to word
    not     $s1,      $s1         # Do ~q | (taking two's complement)
    addi    $s1,      $s1,      1 # Add 1 to q | and getting -q
    add     $s0,      $s0,      $s1 # Add -q to p.
    jr      $ra
```

The screenshot displays the SPIM MIPS simulator interface. On the left, the 'Regs' panel shows the state of various registers, with \$s0 and \$s1 highlighted. The main window is divided into three sections: 'Text Segment' (assembly code), 'Kernel Text Segment', and 'Data Segment'. The 'Text Segment' shows the assembly code for the program, with the 'SUBTRACT' function highlighted. The 'Kernel Text Segment' shows the system call code. The 'Data Segment' shows the memory layout. At the bottom, the 'Output' panel displays the execution results, showing the value 5. The 'Log' panel shows the execution progress, including the start and end of the program.

Assembly code's Output is 5. (Correct: $10 - 5 = 5$)

Q2

Write an assembly language program to find an average of 15 numbers stored at consecutive locations in memory.

C++ Code:

```
float average(int arr[])
{
    int ans = 0;
    int i = 0;
    while(i<15)
    {
        ans += arr[i];
        i++;
    }

    return ans/15.00;
}
```

MIPS Assembly Code:

```

.data
ARRAY: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 # Example array
RESULT: .word 0 # To store the result

.text
.globl main
main:
    la      $s0, ARRAY # Load the base address of the array

    jal     AVERAGE    # Call the Average function

    mov.s   $f12, $f2   # Move result from float register to integer register
    li      $v0, 2       # print float syscall code
    syscall

    # Exit the program
    li $v0, 10
    syscall

AVERAGE: # array address comes in $s0 (as per the function signature)

```

```

    addi    $s1,    $zero,    0    # int ans = 0
    addi    $t0,    $s0,      60    # t0: 15 4 byte numbers ahead of

LOOP:
    beq     $s0,    $t0,      DONE# jump to done if at array end
    lw      $t1,    0($s0)      # load arr[i]
    add     $s1,    $s1,      $t1 # ans += arr[i]
    addi    $s0,    $s0,      4    # i++
    j       LOOP              # loop back to check condition

DONE:
    addi    $t0,    $zero,    15    # t0 = 15
    mtc1    $s1,    $f0          #
    mtc1    $t0,    $f1          #
    cvt.s.w $f0,    $f0          #
    cvt.s.w $f1,    $f1          #
    div.s   $f2,    $f0,        $f1 #
    s.s     $f2,    RESULT      #
    jr      $ra                  # global will find their answer

```

The screenshot displays the SPIM MIPS simulator interface. The main window shows assembly code with comments. The 'Regs' panel on the left lists general and special float registers. The 'Text Segment' panel shows the assembly code with the current instruction highlighted. The 'Data Segment' panel shows memory contents. The 'User Stack' panel shows the stack frame. The 'Output' panel shows the execution speed and a log of events. The 'Log' panel shows the execution progress and a message indicating that the execution finished.

Regs (Hex Dec Bin):

- PC = 0040001c
- EPC = 00000000
- Cause = 00000000
- BadVAddr = 00000000
- Status = 300ff710
- HI = 00000000
- LO = 00000000

General Registers:

- R0 (r0) = 00000000
- R1 (r1) = 00010000
- R2 (v0) = 0000000a
- R3 (v1) = 00000000
- R4 (a0) = 00000000
- R5 (a1) = 7ffff778
- R6 (a2) = 7ffff77c
- R7 (a3) = 00000000
- R8 (t0) = 0000000f
- R9 (t1) = 0000000f
- R10 (t2) = 00000000
- R11 (t3) = 00000000
- R12 (t4) = 00000000
- R13 (t5) = 00000000
- R14 (t6) = 00000000
- R15 (t7) = 00000000
- R16 (s0) = 1001001c
- R17 (s1) = 0000007a
- R18 (s2) = 00000000
- R19 (s3) = 00000000
- R20 (s4) = 00000000
- R21 (s5) = 00000000
- R22 (s6) = 00000000
- R23 (s7) = 00000000
- R24 (t8) = 00000000
- R25 (t9) = 00000000
- R26 (k0) = 00000000
- R27 (k1) = 00000000
- R28 (gp) = 10000000
- R29 (sp) = 7ffff77a
- R30 (s8) = 00000000
- R31 (ra) = 0040002c

Text Segment (Kernel text, Instruction value, Source code):

```

[0040001c] jal 0x00400024 [main] ; 188: jal main
[0040001d] nop ; 189: nop
[0040001e] ori $2, $0, 10 ; 191: li $v0 10
[0040001f] syscall ; 192: syscall # syscall 10 (exit)
[00400020] lui $16, 4097 [ARRAY] ; 8: la $s0, ARRAY # Load the base addr
[00400021] jal 0x00400040 [AVERAGE] ; 10: jal AVERAGE # Call the Average
[00400022] mov.s $f12, $f2 ; 12: mov.s $f12, $f2 #
[00400023] ori $2, $0, 2 ; 13: li $v0, 2 # print float sysca
[00400024] syscall ; 14: syscall
[00400025] ori $2, $0, 10 ; 17: li $v0, 10
[00400026] syscall ; 18: syscall
[00400027] addi $t0, $0, 0 ; 21: addi $t0, $zero, 0 # int ans = 0
[00400028] addi $s0, $16, 60 ; 22: addi $s0, $s0, 60 # t0: 15 4 byte numb
[00400029] beq $s0, $s0, 20 [DONE-0x00400048]; ; 25: beq $s0, $s0, $t0, DONE# jump to done if a
[0040002a] lw $0, 0($s0) ; 26: lw $t1, 0($s0) # load arr[i]
[0040002b] add $17, $17, $0 ; 27: add $s1, $s1, $t1 # ans += arr[i]
[0040002c] addi $s0, $s0, 4 ; 28: addi $s0, $s0, 4 # i++
[0040002d] j 0x00400048 [LOOP] ; 29: j LOOP # loop back to check
[0040002e] addi $s0, $0, 15 ; 32: addi $s0, $zero, 15 # t0 = 15
[0040002f] mtc1 $17, $f0 ; 33: mtc1 $s1, $f0 #
[00400030] mtc1 $t0, $f1 ; 34: mtc1 $t0, $f1 #
[00400031] cvt.s.w $f0, $f0 ; 35: cvt.s.w $f0, $f0 #
[00400032] cvt.s.w $f1, $f1 ; 36: cvt.s.w $f1, $f1 #
[00400033] div.s $f2, $f0, $f1 ; 37: div.s $f2, $f0, $f1 #
[00400034] lui $1, 4097 ; 38: s.s $f2, RESULT #
[00400035] swc1 $f2, 00($s1) ; 39: jr $ra # global will find t
[00400036] jr $s1 ; 39: jr $ra

```

Data Segment (Kernel data, Hex, Dec):

User Data Segment:

```

[10010000] 00000001 00000002 00000003 00000004 .....
[10010010] 00000005 00000006 00000007 00000008 .....
[10010020] 00000009 0000000a 0000000b 0000000c .....
[10010030] 0000000d 0000000e 0000000f 41000000 .....A

```

User Stack (Hex, Dec):

```

[7ffff770] 00000000 00000000 7ffff7ef .....
[7ffff780] 7ffff7e1 7ffff7e2 7ffff7e3 7ffff7e4 .....
[7ffff790] 7ffff7e5 7ffff7e6 00000000 00000000 .....
[7ffff7a0] 3d5f0000 68742f2e 702a7369 72676f72 ...../this.progr
[7ffff7b0] 4c000d01 3d474a41 54552e43 003d2d46 an-LANG-C.UTF-8-
[7ffff7c0] 45464f48 6f682f3d 772f616d 755f6265 HOME=/home/ueb_u
[7ffff7d0] 00720573 3d444570 4150002f 2f3d4054 ser-PWD/-PATH-/
[7ffff7e0] 45535500 65773d52 73755f62 4c007265 -USER=ueb_user-L
[7ffff7f0] 414e474f 773d454d 755f6265 00726573 OGNNAME=ueb_user-

```

Output: 8.00000000

Log: Based on SPIM Version 9.1.20 of August 29, 2017 by James Larus. Execution finished

Code output is 8. (correct)

Single Precision

FG0 = 120.000

FG1 = 15.0000

FG2 = 8.00000

floating point registers state

Q3

Write an assembly language program to find an LCM of two numbers stored at consecutive locations in memory

C++ Code:

```
int lcm(int p, int q)
{
    int temp, a{p}, b{q};
    if(b > a)
    {
        temp = b;
        b = a;
        a = b;
    }

    temp = a % b;
    while(temp != 0)
    {
        a = b;
        b = temp;
        temp = a % b;
    }

    return p * (q / b);
}
```

Assembly Code:

```
.data
num1:    .word    5                # First number
```

```

num2:  .word  3                # Second number
result: .word  0              # To store the result

.text
        .globl  main
main:
    # Load the numbers into registers
    lw      $t0,    num1        # a = num1
    lw      $t1,    num2        # b = num2
    lw      $t2,    result      # ans = 0 (result)

    # Initialize temporary registers
    move     $t3,    $zero      # int i = 0;

multiply:
    beq      $t3,    32,        end_multiply    # while(i < 32){
    andi     $t4,    $t1,    1    # b & 1
    beq      $t4,    $zero,    skip_add         # if(b & 1)
    add      $t2,    $t2,    $t0    # ans += a;

skip_add:
    sll      $t0,    $t0,    1    # a <<= 1;
    srl      $t1,    $t1,    1    # b >>= 1;
    addi     $t3,    $t3,    1    # i++;
    j        multiply            # }

end_multiply:
    # Store the result back to memory
    sw      $t2,    result

    # Exit the program
    li      $v0,    10
    syscall

```

Regs @ Hex ○ Dec □ Bin
□ Kernel text □ Instruction value □ Source code
Data Segment @ Hex ○ Dec □ Bin

Special Registers

PC = 0040003c
EPC = 00000000
BadAddr = 00000000
Status = 300ff1f0
HI = 00000000
LO = 0000000a

General Registers

R0 (r0) =	00000000
R1 (at) =	00000000
R2 (v0) =	0000000a
R3 (v1) =	00000000
R4 (a0) =	0000000a
R5 (a1) =	7ffff7f8
R6 (a2) =	7ffff7fc
R7 (a3) =	00000000
R8 (t0) =	0000000a
R9 (t1) =	00000005
R10 (t2) =	00000000
R11 (t3) =	00000000
R12 (t4) =	00000000
R13 (t5) =	00000000
R14 (t6) =	00000000
R15 (t7) =	00000000
R16 (a8) =	00000000
R17 (s1) =	00000001
R18 (s2) =	00000000
R19 (s3) =	00000000
R20 (s4) =	00000000
R21 (s5) =	00000000
R22 (s6) =	00000000
R23 (t7) =	00000000
R24 (t8) =	00000000
R25 (t9) =	00000000
R26 (a9) =	00000000
R27 (x1) =	00000000
R28 (d0) =	10010000
R29 (sp) =	7ffff7f4
R30 (a10) =	00000000
R31 (ra) =	0040002c

Text Segment

```

[00400004] addiu $5, $20, 4      ; 184: addiu $a1 $p4 # argv
[00400008] addiu $6, $5, 4        ; 185: addiu $a2 $a1 4 # envp
[0040000c] sll $2, $4, 2         ; 186: sll $v0 $w0 2
[00400010] addu $6, $6, $2        ; 187: addu $a2 $a2 $v0
[00400014] jal 0x0040002a [main] ; 188: jal main
[00400018] nop                 ; 189: nop
[0040001c] ori $2, $0, $10       ; 191: li $v0 $10
[00400020] syscall              ; 192: syscall          # syscall 10 (exit)
[00400024] lui $28, 4097 [PQ]   ; 7: la    $gp, PQ      # Load address of pq
[00400028] jal 0x0040000a [CM]   ; 10: jal   CM          # Call the loc fnc
[00400032] addi $4, $16, 0       ; 11: addi  $a0, $s0, 0  # store output for p
[00400038] ori $2, $16, 1       ; 12: li    $v0, 1     # print integer code
[0040003c] syscall              ; 13: syscall         # print output
[00400038] ori $2, $0, $10      ; 16: li    $v0, $10
[00400040] syscall              ; 17: syscall
[00400044] lw $16, 0($28)        ; 20: lw    $s0, 0($gp)  # Load p
[00400044] lw $17, 4($28)        ; 21: lw    $s1, 4($gp)  # Load q
[00400048] sub $10, $10, $17     ; 23: sub   $t2, $s0, $s1 # calculate p - q
[0040004c] addi $8, $17, 0       ; 24: addi  $t0, $s1, 0  # a = a * q
[00400050] addi $9, $16, 0       ; 25: addi  $t1, $s0, 0  # b = b * p
[00400054] bltz $8, $12 [LOOP-0x00400054] ; 26: bltz  $t2, LOOP    # jump to loop if p
[00400054] blaz $8, $16, 0       ; 27: addi  $t0, $s0, 0  # a = p
[0040005c] addi $9, $17, 0       ; 28: addi  $t1, $s1, 0  # b = q
[00400060] div $8, $9           ; 31: div    $t0, $t1     # HI = a % b
[00400064] mhu $10             ; 32: mfhi   $t2          # temp = HI
[00400068] beq $10, $0, 16 [DONE-0x00400068]; 33: beqz   $t2, DONE    # jump to done if t
[0040006c] addi $9, $9, 0       ; 34: addi  $t1, $s1, 0  # a = b
[00400070] mhu $9              ; 35: mfhi   $t1          # b = HI
[00400074] j 0x00400060 [LOOP] ; 36: j      LOOP        # loop back to check
[00400078] div $17, $9          ; 39: div    $s1, $t1     # LO = q / b
[0040007c] mlo $17             ; 40: mlo    $s1          # q = LO
[00400080] mul $16, $16, $17     ; 41: mul    $s0, $s0, $s1 # $s1 HI, LO = p * q
[00400084] sw $16, 0($28)        ; 42: sw     $s0, 0($gp)  # store t0(LO) in lo
[00400088] jr $31               ; 43: j      $ra

```

Execution speed: ● Play Step Reset Click line to toggle breakpoint

Output

10

Log

Based on SPIN Version 9.1.10 of August 29, 2017
by James Larus.

Execution finished

User Data Segment

[10010000] 0000000a 00000005 0000000a 00000000

User Stack

@ Hex ○ Dec □ Bin

```

[7ffffff0] 00000000 00000000 7fffffff .....
[7ffffff0] 7fffffff 7fffffff 7fffffff 7fffffff
[7ffffff0] 7fffffff 7fffffff 7fffffff 00000000
[7ffffff0] 3d5f0000 687af2fe 79e27369 72676772 .../.:/this_prog
[7ffffff0] ac0b0651 3d474ae1 54552ea3 00382d46 am-LANG-C.UTF-8
[7ffffff0] 454d4f48 6f682f3d 727265fd 75562625 HOME=/home/web_u
[7ffffff0] 00726573 3d445750 4158082f 2f3d4854 ser-Pad/-PATH/

```

```

num2:  .word  3                # Second number
result: .word  0              # To store the result

.text
.globl  main
main:
    # Load the numbers into registers
    lw    $t0,    num1        # a = num1
    lw    $t1,    num2        # b = num2
    lw    $t2,    result      # ans = 0 (result)

    # Initialize temporary registers
    move   $t3,    $zero      # int i = 0;

multiply:
    beq    $t3,    32,    end_multiply  # while(i < 32){
    andi   $t4,    $t1,    1            # b & 1
    beq    $t4,    $zero,    skip_add    # if(b & 1)
    add    $t2,    $t2,    $t0          # ans += a;

skip_add:
    sll    $t0,    $t0,    1            # a <<= 1;
    srl    $t1,    $t1,    1            # b >>= 1;
    addi   $t3,    $t3,    1            # i++;
    j      multiply                    # }

end_multiply:
    # Store the result back to memory
    sw     $t2,    result

    addi   $a0,    $t2,    0            # load value into a0
    li     $v0,    1                    # print integer
    syscall

    # Exit the program
    li     $v0,    10
    syscall

```


[illegible]

```

    return {2, -1, -1};
}

```

Assembly Code

```

.data
list: .word 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 # Sorted list of 10 r
X:    .word 7, 0, -1, -1                    # Given number to fir

.text
.globl main
main:
    la    $s0,    list                    # Load base address of t
    lw    $s1,    X                      #
    la    $gp,    X                      # global pointer at X's
    jal   findNum

    lw    $a0,    4($gp)                  # load the output for pr
    li    $v0,    1                      #
    syscall

    li    $v0,    10
    syscall

findNum:
    li    $t0,    0                      # iterations = 0
    li    $t1,    0                      # left = 0
    li    $t2,    9                      # right = 9

loop:
    sub    $t3,    $t1,                    $t2    # left - right
    bgtz   $t3,    endLoop                # end loop if left > rig
    addi   $t0,    1                      # iterations++
    add    $t4,    $t1,                    $t2    # mid = left + right
    srl    $t4,    $t4,                    1      # mid >= 1
    sll    $t5,    $t4,                    2      # t5 = mid*4
    add    $t5,    $s0,                    $t5    # t5 = &list + 4*(mid)
    lw     $t5,    0($t5)                  # load arr[mid]

    beq    $t5,    $s1,                    found  # found if arr[mid] = X
    sub    $t5,    $t5,                    $s1    # arr[mid] - X
    bltz   $t5,    updateLeft              # update_left if arr[mic

    addi   $t2,    $t4,                    -1     # right = mid - 1
    j      loop

```

```

updateLeft:
    addi    $t1,    $t4,            1        # left = mid + 1
    j       loop

endLoop:
    li      $t0,    2
    sw      $t0,    4($gp)              # store 2 at output (right)
    j       $ra

found:
    li      $t1,    1
    sw      $t1,    4($gp)              # store 1 at output
    sw      $t0,    8($gp)              # store iterations
    sw      $t4,    12($gp)             # store index of X
    j       $ra

```

The screenshot displays the SPIM MIPS simulator interface. The main window shows assembly code with comments. The left sidebar lists special and general registers. The right sidebar shows the data segment and user stack. The bottom section includes execution controls and an output/log window.

Assembly Code:

```

[00400030] lui $1, 4097 [X] ; 10: la $gp, X # global poi
[00400034] ori $20, $1, 40 [X] ; 11: jal findNum
[00400038] jal 0x00400050 [findNum] ; 13: lw $a0, 4($gp) # load the o
[00400040] lw $4, 4($20) ; 14: li $v0, 1 #
[00400044] syscall ; 15: syscall
[00400048] ori $2, $0, 10 ; 17: li $v0, 10
[00400050] syscall ; 18: syscall
[00400050] ori $0, $0, 0 ; 21: li $t0, 0 # iterations
[00400054] ori $0, $0, 0 ; 22: li $t1, 0 # left = 0
[00400058] ori $10, $0, 9 ; 23: li $t2, 9 # right = 9
[0040006c] sub $11, $9, $10 ; 26: sub $t3, $t1, $t2 # left - rig
[0040006e] bgtz $11 $0 [endLoop-0x0040006e]; 27: bgtz $t3, endLoop # end loop i
[00400064] addi $0, $0, 1 ; 28: addi $t0, 1 # iterations
[00400068] add $12, $9, $10 ; 29: add $t4, $t1, $t2 # mid = left
[0040006c] srl $12, $12, 1 ; 30: srl $t4, $t4, 1 # mid >>= 1
[00400070] sll $13, $12, 2 ; 31: sll $t5, $t4, 2 # t5 = mid*4
[00400074] add $13, $10, $13 ; 32: add $t5, $t0, $t5 # t5 = &list
[00400078] lw $13, 0($13) ; 33: lw $t5, 0($t5) # load arr[m
[0040007c] beq $13, $17, 40 [found-0x0040007c]; 35: beq $t5, $t1, found # found i
[00400080] sub $13, $13, $17 ; 36: sub $t5, $t5, $t1 # arr[mid] -
[00400084] bltz $13 12 [updateLeft-0x00400084]; 37: bltz $t5, updateLeft # update_
[00400088] addi $10, $12, -1 ; 39: addi $t2, $t4, -1 # right = ml
[00400090] j 0x0040005c [loop] ; 40: j loop
[00400094] addi $0, $12, 1 ; 43: addi $t1, $t4, 1 # left = mid
[00400098] j 0x0040005c [loop] ; 44: j loop
[0040009c] ori $0, $0, 2 ; 47: li $t0, 2
[0040009e] sw $0, 4($20) ; 48: sw $t0, 4($gp) # store 2 at
[004000a0] jr $31 ; 49: j $ra
[004000a4] ori $0, $0, 1 ; 52: li $t1, 1
[004000a8] sw $0, 4($20) ; 53: sw $t1, 4($gp) # store 1 at
[004000ac] sw $0, 8($20) ; 54: sw $t0, 8($gp) # store iter
[004000b0] sw $12, 12($20) ; 55: sw $t4, 12($gp) # store inde
[004000b4] jr $31 ; 56: j $ra

```

Registers:

- Special Registers: PC = 0040003c, EPC = 00000000, Cause = 00000000, BadVAddr = 00000000, Status = 3000ff10, HI = 00000000, LO = 00000000.
- General Registers: R0 (r0) = 00000000, R1 (at) = 10010000, R2 (v0) = 00000000, R3 (v1) = 00000000, R4 (a0) = 00000001, R5 (a1) = 7ffffff8, R6 (a2) = 7ffffffc, R7 (a3) = 00000000, R8 (t0) = 00000004, R9 (t1) = 00000001, R10 (t2) = 00000003, R11 (t3) = 00000000, R12 (t4) = 00000003, R13 (t5) = 00000007, R14 (t6) = 00000000, R15 (t7) = 00000000, R16 (s0) = 10010000, R17 (s1) = 00000007, R18 (s2) = 00000000, R19 (s3) = 00000000, R20 (s4) = 00000000, R21 (s5) = 00000000, R22 (s6) = 00000000, R23 (s7) = 00000000, R24 (t8) = 00000000, R25 (t9) = 00000000, R26 (s0) = 00000000, R27 (k1) = 00000000, R28 (gp) = 10010028, R29 (sp) = 7ffffff4, R30 (s8) = 00000000, R31 (ra) = 0040003c.

Data Segment:

User Data Segment

```

[10010000] 00000001 00000003 00000005 00000007 .....
[10010010] 00000009 0000000b 0000000d 0000000f .....
[10010020] 00000011 00000013 00000007 00000001 .....
[10010030] 00000004 00000003 00000000 00000000 .....

```

User Stack:

```

[7ffffff0] 00000000 00000000 7ffffffe .....
[7ffffff8] 7ffffffe 7ffffffd 7ffffff4 7ffffffc .....
[7ffffff0] 7ffffff3 7ffffff2 00000000 00000000 .....
[7ffffff0] 3d5f0000 6874222e 702e7369 72676f72 - ./this.progn
[7ffffff0] 4c065061 3d474e41 54524433 00302046 an-LANG-C.UTF-8
[7ffffff0] 45444f48 6f682f3d 772f656d 755f6265 HOME=/home/web_u
[7ffffff0] 00726573 3d445750 4150002f 2f3d4854 ser-PID=/-PATH=

```

output as 1. 4 iterations, and 3 index stored in data segment.

Q6

Write an assembly language program to find a character in a string.

C++ Code:

```

int findChar(char arr[], char c, int arrLength)
{
    // function returns -1 if not found, otherwise
    // returns index of c in string
}

```

```

int i = 0;
while(i < arrLength)
{
    if(arr[i] == c) return i;
    i++;
}
return -1;
}

```

Assembly Code:

```

.data
string:      .asciiz "Finding a Character"      # Example string
char_to_find: .byte  'a'                      # c = 'a'
string_length: .word  11
result:      .word  0

.text
.globl  main

main:
    la    $s0,      string                    # $s0 = arr[]
    lb    $s1,      char_to_find              # $s1 = c
    lw    $s2,      string_length             # $s2 = arrLength
    la    $s3,      result                    # address to store result
    jal   findChar

    addi   $a0,      $t0,      0               #
    li     $v0,      1                       #
    syscall

    li     $v0,      10                      # Exiting the program
    syscall

findChar:
    move   $t0,      $s0                     # int i = 0
    add    $t1,      $t0,      $s2           # define array end

loop:
    beq    $t0,      $t1,      not_found     # end if i==arrLength
    lb     $t2,      0($t0)                  # load arr[i]
    beq    $t2,      $s1,      found         # if arr[i] == c
    addi   $t0,      $t0,      1             # i++
    j      loop
not_found:
found:

```

```
not_found:
    li    $t0,    -1                # ans = -1
    j     write_back

found:
    sub   $t0,    $t0,    $s0       # ans = &arr[i]

write_back:
    sw    $t0,    0($s3)            # return ans
    j     $ra
```

Regs
Hex Dec Bin

Special Registers

PC = 00400054
 EPC = 00000000
 Cause = 00000000
 BadVAddr = 00000000
 Status = 10000F10
 HI = 00000000
 LO = 00000000

General Registers

R0 (r0)	= 00000000
R1 (at)	= 10010000
R2 (v0)	= 0000000A
R3 (v1)	= 00000000
R4 (a0)	= 0000000C
R5 (a1)	= 7FFFFFFB
R6 (a2)	= 7FFFFFFC
R7 (a3)	= 00000000
R8 (t0)	= 00000000
R9 (t1)	= 10010000
R10 (t2)	= 00000001
R11 (t3)	= 00000000
R12 (t4)	= 00000000
R13 (t5)	= 00000000
R14 (t6)	= 00000000
R15 (t7)	= 00000000
R16 (a0)	= 10010000
R17 (s1)	= 00000001
R18 (s2)	= 00000000
R19 (s3)	= 1001001C
R20 (s4)	= 00000000
R21 (s5)	= 00000000
R22 (s6)	= 00000000
R23 (s7)	= 00000000
R24 (t8)	= 00000000
R25 (t9)	= 00000000
R26 (a0)	= 00000000
R27 (x1)	= 00000000
R28 (g0)	= 10001000
R29 (fp)	= 7FFFFFF4
R30 (s8)	= 00000000
R31 (ra)	= 00400044

Text Segment

```

[00400010] addu $6, $6, $2          ; 187: addu $a2 $a2 $v0
[00400014] jal 0x00400024 [main] ; 188: jal main
[00400018] nop                  ; 189: nop
[00400022] ori $2, $0, 10       ; 191: li $v0 10
[00400026] syscall              ; 192: syscall # syscall 10 (exit)
[0040002A] lui $16, 4097 [string]; 11: la $s0, string # $s
[0040002E] lui $1, 4097         ; 12: lb $s1, char_to_find # $s
[00400032] lb $17, 20($s1)      ; 13: lw $s2, string_length # $s
[00400036] lui $18, 24($s1)    ; 14: la $s3, result # ad
[0040003A] lui $1, 4097 [result]; 
[0040003E] ori $19, $1, 20 [result]; 
[00400042] jal 0x00400058 [findChar]; 
[00400046] addi $4, $8, 0       ; 15: jal findChar
[0040004A] ori $2, $0, 1        ; 17: addi $a0, $t0, 0 #
[0040004E] syscall             ; 18: li $v0, 1 #
[00400052] syscall             ; 19: syscall
[00400056] ori $2, $0, 10      ; 21: li $v0, 10 # Ex
[0040005A] syscall            ; 22: syscall
[0040005E] addu $8, $0, $16     ; 25: move $t0, $t0 # In
[00400062] addu $8, $0, $10     ; 26: addu $t1, $t0, $s2 # de
[00400066] beq $8, $9, 20 [not_found-0x00400060]; 29: beq $t0, $t1, not_found #
[0040006A] lb $10, 0($8)        ; 30: lb $t2, 0($t0) # 10
[0040006E] beq $10, $17, 24 [found-0x00400068]; 31: beq $t2, $s1, found #
[00400072] addi $8, $8, 1       ; 32: addi $t0, $t0, 1 # 1+
[00400076] j 0x00400050 [loop]  ; 33: j loop
[0040007A] lui $1, -1           ; 36: li $t0, -1 # an
[0040007E] ori $8, $1, -1       ; 
[00400082] j 0x00400054 [write_back]; 37: j write_back
[00400086] sub $8, $8, $16      ; 40: sub $t0, $t0, $s0 # an
[0040008A] sw $8, 0($19)        ; 43: sw $t0, 0($s3) # re
[0040008E] jr $31               ; 44: jr $ra
      
```

Execution speed: ● Play Step Reset Click line to toggle breakpoint

Output

Log

B Based on SPH9 Version 9.1.20 of August 29, 2017 by James Larus

Execution Finished

Data Segment
Kernal Data Hex Dec Charac

User Data Segment

```

[10010000] 646e694 20576e69 68432061 63617261 Finding a Character
[10010010] 00726574 00000061 00000000 00000008 ter-a.....
      
```

User Stack
Hex Dec

```

[7ffffff0] 00000000 00000000 7fffffff .....
[7ffffff8] 7fffffff 7fffffff 7fffffff .....
[7ffffffc] 7fffffff 7fffffff 00000000 00000000 .....
[7ffffff4] 3d5f0000 08742f2e 702a7309 72fe67f2 .../,this,prog
[7ffffff8] AC006d51 3d474a51 54552da3 00382546 an-LANG-C.UTF-8-
[7ffffffc] A54d4f48 0fb2f234 72f1556d 755f6265 HDR-/home/web_u
[7ffffff0] 0072b573 3d454570 4150802f 2f3d4545 ser-PAD-/PATH/
[7ffffff8] A5C35200 657345c9 7375c64e 4e007405 HTER-unsh-res
      
```

Output for the code (8 as index of first occurrence of a)