2016

Computer Architecture Lab

LAB 2: CONTROL FLOW (INCLUDING LOOPS AND BRANCHES), INPUTS & OUTPUTS, MAKING DECISIONS

JEFFREY HUANG | RUID : 159-00-4687

Code used for Assignment 1

```
# Jeffrey Huang
# RUID: 159004687
# NETID: jh1127
# branch.asm - Loop and Branches Program
# Registers used:
# $t0 - used to hold the number of iteration
# $t1 - used to hold the counter value in each iteration
# $v0 - syscall parameter and return value.
# $a0 - syscall parameter -- the string to print.

.text

main:
    li $t0, 5                   # init the number of loop.
    li $t1, 0                   # init the counter of the loop

loop:
    beq $t0, $t1, endloop       # if $t0 == $t1 then go to endloop
    addi $t1, $t1, 1            # increment the counter, i.e. $t1++
    b loop                      # branch to loop label

endloop:
    li $v0, 10                  # 10 is the exit syscall.
    syscall

# Note for Assignment 1
# "beq $t0, $t1, endloop" -> the loop will execute 6 times
#   Reason: branch to endloop if $t0 == $t1
# "blt $t0, $t1, endloop" -> the loop will execute 7 times
#   Reason: branch to endloop if $t0 < $t1
# "bge $t0, $t1, endloop" -> the loop will execute 0 times
#   Reason: branch to endloop if $t0 >= $t1
# "bnez $t1, endloop" -> the loop will execute 1 time
#   Reason: branch to endloop if $t1 != 0
```

Code used for Assignment 2

```
# Jeffrey Huang
# RUID: 159004687
# NETID: jh1127
# Assignment 2

# Pseudocode:   The procedure of the code for finding the largest prime number under 300 is easier to be brute
#               forced than other methods that involve finding a large prime number. If i need to find multiple
#               prime numbers then a different method would be more efficient. However, given the condition
#               that the target is going to be less than 300. I primarily found the largest prime number under
#               300 to be 293. Hence the procedure for finding the prime number involves two loops. The first
#               loop involves checking the remainder of the number. If the remainder is equal to 0, then the
#               number is not a prime number. As a result, we don't continue checking the number and then
#               proceed to check the next number. The second loop is check the prime number condition
#               then if the condition is not met, the number that is being checked will be decremented.
#               This process will repeat until the remainder condition is never 0 and then the program will
#               print out the prime number in console.

.text

main:
    li $t0, 299
    li $t1, 2

loop:
    beq $t0, $t1, endloop       # end condition is where if the remainder is the same as the number being checked.
    rem $t2, $t0, $t1           # finds the remainder of the number.
    beq $t2, $0, decrement      # If during any time the reaminder  is equal to 0, the number is NOT a prime and
                                # decremented.
    add $t1, 1                  # Current divisor does not result in a remainder with the current number. Check
                                # a different divisor.
    j loop                      # jumps to loop

decrement:
    sub $t0, $t0, 1             # decrements the current checking number by 1
    li $t1, 2                   # changes the divisor back to the initial value of 2
    j loop                      # jumps back to loop

endloop:
    li $v0, 1                   # setting the syscall argument to print integer
    move $a0, $t0               # moving the prime number to the argument register
    syscall                     # syscall to print the prime number
    li $v0, 10                  # setting the syscall argument to end the program
    syscall                     # syscall to end the program;
```

Output for Assignment 2

Console

```
293
```

Code used for Assignment 3

```
# Jeffrey Huang
# RUID: 159004687
# NETID: jh1127
# Assignment 3

# Pseudocode:   (1) Get user inputs for the first and second number. -> $t0 = a; $t1 = b;
#               (2) check if the numbers are in bounds. -> $t2 = 10; $t3 = 500;
#                   return error if false
#               (3) check if $t0 is divisable by $t1, return error if true
#               (4) check if $t1 is divisable by $t0, return error if true
#               (5) order the two input numbers are use them as counters
#               (6) if the lower number is odd, add 1 then increment by 2 until
#                   counter is greater than or equal to the larger number inputted
#                   if the lower number is odd, increment by 2 until the counter is
#                   greater than or equal to the larger number inputted
#               (7) Check if number is even, if even, add to sum then increment by 2
#                   Check if number is even, if odd, increment number by 1 to make even.
#               (8) Print out sum


.data
    str1: .asciiz "Enter the first number: "
    str2: .asciiz "Enter the second number: "
    errorStr: .asciiz "Error: numbers are divisable by one another or not in the range of 10 ~ 500"

.text
main:
## Get first number from user, put into $t0.
    li $v0, 4               # load syscall print string
    la $a0, str1            # load address of str1 to register a0
    syscall                 # make the syscall.
    li $v0, 5               # load syscall read_int into $v0.
    syscall                 # make the syscall.
    move $t0, $v0           # move the number read into $t0.

## Get second number from user, put into $t1.
    li $v0, 4               # load syscall print string
    la $a0, str2            # load address of str2 to register a0
    syscall                 # make the syscall.
    li $v0, 5               # load syscall read_int into $v0.
    syscall                 # make the syscall.
    move $t1, $v0           # move the number read into $t1.

## Condition: Checking bounds
    li $t2, 10              # loading lower bound to 10
    li $t3, 500             # loading upper bound to 400
    blt $t0, $t2, error     # branch if first input is less than 10
    bgt $t0, $t3, error     # branch if first input is greater than 500
    blt $t1, $t2, error     # branch if second input is less than 10
    bgt $t1, $t3, error     # branch if second input is greater than 500
## Condition: Checking divisibility
    rem $t2, $t0, $t1       # finding the remainder of $t0 / $t1
    beq $t2, $0, error      # If remainder is 0, then first input is divisable by the second input
    rem $t2, $t1, $t0       # finding the remainder of $t1 / $t0
    beq $t2, $0, error      # If remainder is 0, then second input is divisable by the first input

## Condition: Check which number is larger
    bgt $t0, $t1, swap      # If $t0 > $t1, then swap values
    move $t4, $t0           # $t4 will be used as a counter register
    addi $t4, $t4, 1        # Two inputs are exclusive in the calculations
    li $t5, 2               # Used for remainder check
```

```
loop:
    bge $t4, $t1, exit      # If $t4 >= $t1, then exit
    rem $t6, $t4, $t5       # Find whether the number is divisable by 2
    beqz $t6, even          # if remainder == 0, then the number is even
    bnez $t6, odd           # if remainder != 0, then the number is odd
    j loop                  # jump to loop

even:
    add $t7, $t7, $t4       # number is even, add to sum
    addi $t4, $t4, 2        # increment number by 2 to get next even number
    j loop                  # jump to loop

odd:
    addi $t4, $t4, 1        # number is even, increment by 1
    j loop                  # jump to loop

swap:
    move $t2, $t1           # $t2 temporary place holder for the value to be swapped
    move $t1, $t0           # moving $t0 to $t1
    move $t0, $t2           # moving $t2 to $t0
    j loop                  # jump to loop

error:
    li $v0, 4               # setting the syscall argument to print string
    la $a0, errorStr        # loading the string to the argument register
    syscall                 # syscall to print out the string
    li $v0, 10              # setting the syscall argument to end the program
    syscall                 # syscall to end the program

exit:
    li $v0, 1               # setting the syscall argument to print integer
    move $a0, $t7           # moving the sum to the argument register
    syscall                 # syscall to print out the integer
    li $v0, 10              # setting the syscall argument to end the program
    syscall                 # syscall to end the program
```

Output for Assignment 3

Console

```
Enter the first number: 51
Enter the second number: 101
1900
```

Console

```
Enter the first number: 5
Enter the second number: 14
Error: numbers are divisable by one another or not in the range of 10 ~ 500
```

Console

```
Enter the first number: 2
Enter the second number: 1009
Error: numbers are divisable by one another or not in the range of 10 ~ 500
```

Console

```
Enter the first number: 109
Enter the second number: 499
59280
```

Pseudo-code used for Assignment 4

```
Scanner input = new Scanner(System.in);
System.out.println("The number of lines? ");
int lines = input.nextInt();
int half = (lines + 1) / 2;
int i;
for(i = 1; i <= half; i++){
    int spaces = half - i;
    int stars = 2 * i - 1;
    while(spaces > 0){
        System.out.print(".");
        spaces--;
    }
    while(stars > 0){
        System.out.print("*");
        stars--;
    }
    System.out.print("\n");
}
if(lines % 2 == 0){
    for(i = i - 1; i > 0; i--){
        int spaces = half - i;
        int stars = 2 * i - 1;
        while(spaces > 0){
            System.out.print(".");
            spaces--;
        }
        while(stars > 0){
            System.out.print("*");
            stars--;
        }
        System.out.print("\n");
    }
}else{
    for(i = i - 2; i > 0; i--){
        int spaces = half - i;
        int stars = 2 * i - 1;
        while(spaces > 0){
            System.out.print(".");
            spaces--;
        }
        while(stars > 0){
            System.out.print("*");
            stars--;
        }
        System.out.print("\n");
    }
}
```

Code used for Assignment 4

```
# Jeffrey Huang
# RUID: 159004687
# NETID: jh1127
# Assignment 4

# Registers:    $t0 -> n -> number of lines to be printed
#               $t1 -> h -> half the number of lines
#               $t2 -> s -> number of spaces to be printed
#               $t3 -> a -> number of asterisks to be printed
#               $t4 ->
#               $t5 ->
#               $t6 -> remainder from checker
#               $t7 -> number for even/odd checker
#               $s0 -> i -> counter

.data
    input: .asciiz "The number of lines? "
    asterisk: .asciiz "*"
    space: .asciiz " "
    newLine: .asciiz "\n"

.text

main:
## Initializing the base values
    li $s0, 1                   # loading initial value of i
    li $t7, 2                   # loading initial value of 2


## Get number of lines from user
    li $v0, 4                   # load syscall service print_string
    la $a0, input               # load syscall argument for print_string
    syscall                     # make syscall to print_string
    li $v0, 5                   # load syscall service read_int
    syscall                     # make syscall to read_int
    move $t0, $v0               # move number read into register $t0
    beqz $t0, exit              # if $t0 == 0, branch to exit
    addi $t1, $t0, 1            # n + 1
    div $t1, $t1, 2             # (n + 1) / 2
    j topLoop                   # jump to topLoop

topLoop:
    bgt $s0, $t1, adjustment    # if i > half, branch to adjustment
    sub $t2, $t1, $s0           # (h - i) -> number of spaces
    mul $t3, $s0, 2             # (2 * i)
    sub $t3, $t3, 1             # (2 * i) - 1 -> number of asterisks
    j printSpacesTop            # jump to printSpacesTop

adjustment:
    rem $t6, $t0, $t7           # i % 2
    sub $s0, $s0, 1             # i = i - 1
    beqz $t6, botLoop           # i % 2 == 0
    sub $s0, $s0, 1             # i = i - 1
    bnez $t6, botLoop           # i % 2 != 0
```

```
botLoop:
    beqz $s0, exit              # if i == 0, branch to adjustment
    sub $t2, $t1, $s0           # (h - i) -> number of spaces
    mul $t3, $s0, 2             # (2 * i)
    sub $t3, $t3, 1             # (2 * i) - 1 -> number of asterisks
    j printSpacesBot

printSpacesTop:
    beqz $t2, printStarsTop     # if $t2 == 0, branch to printStars
    li $v0,4                    # load syscall service print_string
    la $a0, space               # load syscall argument for print_string
    syscall                     # make syscall to print_string
    sub $t2, $t2, 1             # $t2--;
    j printSpacesTop            # jump to printSpacesTop

printStarsTop:
    beqz $t3, printNewLineTop   # if $t3 == 0, branch to printStars
    li $v0,4                    # load syscall service print_string
    la $a0, asterisk            # load syscall argument for print_string
    syscall                     # make syscall to print_string
    sub $t3, $t3, 1             # $t3--;
    j printStarsTop             # jump to printStarsTop

printNewLineTop:
    li $v0,4                    # load syscall service print_string
    la $a0, newLine             # load syscall argument for print_string
    syscall                     # make syscall to print_string
    addi $s0, $s0, 1            # i++
    j topLoop                   # jump to topLoop

printSpacesBot:
    beqz $t2, printStarsBot     # if $t2 == 0, branch to printStars
    li $v0,4                    # load syscall service print_string
    la $a0, space               # load syscall argument for print_string
    syscall                     # make syscall to print_string
    sub $t2, $t2, 1             # $t2--;
    j printSpacesBot            # jump to printSpacesBot

printStarsBot:
    beqz $t3, printNewLineBot   # if $t3 == 0, branch to printStars
    li $v0,4                    # load syscall service print_string
    la $a0, asterisk            # load syscall argument for print_string
    syscall                     # make syscall to print_string
    sub $t3, $t3, 1             # $t3--;
    j printStarsBot             # jump to printStarsBot

printNewLineBot:
    li $v0,4                    # load syscall service print_string
    la $a0, newLine             # load syscall argument for print_string
    syscall                     # make syscall to print_string
    sub $s0, $s0, 1             # i--
    j botLoop                   # jump to botLoop

exit:
    li $v0, 10                  # load syscall service end_program
    syscall                     # make syscall to end program
```

Output for Assignment 4

Console

```
The number of lines? 5
  *
 ***
*****
 ***
  *
```

Console

```
The number of lines? 3
 *
***
 *
```

Console

```
The number of lines? 4
 *
***
***
 *
```

Console

```
The number of lines? 27
                *
               ***
              *****
             *******
            *********
           ***********
          *************
         ***************
        *****************
       *******************
      *********************
     ***********************
    *************************
   ***************************
  *****************************
*******************************
  *****************************
   ***************************
    *************************
     ***********************
      *********************
       *******************
        *****************
         ***************
          *************
           ***********
            *********
             *******
              *****
               ***
                *
```

Console

```
The number of lines? 8
    *
   ***
  *****
 *******
 *******
  *****
   ***
    *
```

Pseudo-code used for Assignment 5

```
public static int numOfTries = 2;

public static void main(String[] args) {
    getPassword1();
}

public static void getPassword1(){
    System.out.print("Set a password: ");
    password1 = input.next();
    checkPassword1();
}

public static void checkPassword1(){
    if(password1.length() < 6 || password1.length() > 10){
        System.out.println("Failed. Please enter a password with the size of 6 to 10");
        getPassword1();
    }else{
        getPassword2();
    }
}

public static void getPassword2(){
    System.out.print("Re-enter the password: " );
    password2 = input.next();
    checkPassword2();
}

public static void checkPassword2(){
    if((!password2.equals(password1)) && numOfTries > 0){
        numOfTries--;
        System.out.print("Incorrect, you have " + numOfTries + " more chance! Please re-enter the password: ");
        getPassword2();
    }else if(numOfTries == 0){
        System.out.println("Incorrect, you have 0 more chance!");
        System.out.println("Password setup failed.");
    }
    else{
        System.out.println("Password is setup");
    }
}
```

Code used for Assignment 5

```
# Jeffrey Huang
# RUID: 159004687
# NETID: jh1127
# Assignment 5

# Registers:    $t0 -> 1st input password
#               $t1 -> length of 1st password
#               $t2 -> temporary byte holder for $t1
#               $t3 -> 2nd input password
#               $t4 -> temporary byte holder for $t3
#               $t5 -> copy of $t1
#               $t6 -> copy of $t3
#               $t7 -> number of tries remaining
#               $s0 -> password minimum length = 6
#               $s1 -> password maximum length = 10
#               $s2 -> constant 2
#               $s3 -> contant 1

.data
    blank:      .space      100
    blank2:     .space      100
    setPass:    .asciiz     "Set a password: "
    badPass:    .asciiz     "Failed. Please enter a password with the size of 6 to 10!\n"
    retryPass:  .asciiz     "Re-enter the password: "
    goodPass:   .asciiz     "Password is setup."
    twoTry:     .asciiz     "Incorrect, you have 2 more chances! Please re-enter the password: "
    oneTry:     .asciiz     "Incorrect, you have 1 more chance! Please re-enter the password: "
    noTry:      .asciiz     "Incorrect, you have no more chances! Password setup has failed."

.text

main:
# Initializing all the values needed.
    li $s0, 6                   # min length = 6
    li $s1, 10                  # max length = 10
    li $s2, 2                   # tries remaining -> 2
    li $s3, 1                   # tries remaining -> 1
    li $t7, 3                   # number of tries remaining = 3
    j getPassword1              # jump to getPassword1

getPassword1:
    li $t1, 0                   # setting length = 0
    li $v0, 4                   # load syscall service print_string
    la $a0, setPass             # load syscall argument for print_string
    syscall                     # make syscall to print_string
    li $v0, 8                   # load syscall service read_string
    la $a0, blank               # load syscall argument for read_string
    syscall                     # make syscall to read_string
    move $t0, $a0               # moving user input into register $t0
    move $t5, $t0               # copy of address
    j getLength1                # jump to getLength1
```

```
getPassword2:
    li $v0, 4                   # load syscall service print_string
    la $a0, retryPass           # load syscall argument for print_string
    syscall                     # make syscall to print_string
    li $v0, 8                   # load syscall service read_string
    la $a0, blank2              # load syscall argument for read_string
    syscall                     # make syscall to read_string
    move $t3, $a0               # moving user input into register $t0
    move $t6, $t3               # copy of address
    j checkPassword2            # jump to checkPassword2

getLength1:
    lb $t2, 0($t0)              # load the first byte from address in $t0
    beqz $t2, checkPassword1    # if $t2 == 0, then go to checkPassword1
    add $t0, 1                  # else increment the address
    add $t1, $t1, 1             # incremenent the counter
    j getLength1                # jump to getLength1

checkPassword1:
    sub $t0, $t0, $t1           # returning the address back to the original location
    sub $t1, $t1, 1             # adjustment so the end byte isn't counted
    blt $t1, $s0, passError     # if length < 6, return error and loop
    bgt $t1, $s1, passError     # if length > 10, return error and loop
    j getPassword2              # jump to getPassword2

checkPassword2:
    lb $t2, 0($t0)              # load the first byte from address in $t0
    lb $t4, 0($t3)              # load the first byte from address in $t3
    bne $t2, $t4, retryError    # if A[i] != B[i], return error and loop
    add $t0, 1                  # else increment the address
    add $t3, 1                  # else increment the address
    sub $t1, $t1, 1             # length--
    beqz $t1, goodSetup         # if nothing is bad go to goodSetup
    j checkPassword2            # jump to checkPassword2

passError:
    li $v0, 4                   # load syscall service print_string
    la $a0, badPass             # load syscall argument for print_string
    syscall                     # make syscall to print_string
    j getPassword1              # jump to getPassword1

retryError:
    addi $t7, $t7, -1           # numOfTries--
    move $t0, $t5                   # restarting address
    move $t3, $t6                   # restarting address
    beq $t7, $s2, twoTriesLeft  # if $t7 == 2, go to twoTriesLeft
    beq $t7, $s3, oneTryLeft    # if $t7 == 1, go to oneTryLeft
    beqz $t7, noTriesLeft       # if $t7 == 0, go to noTriesLeft

twoTriesLeft:
    li $v0, 4                   # load syscall service print_string
    la $a0, twoTry              # load syscall argument for print_string
    syscall                     # make syscall to print_string
    j getPassword2              # jump to getPassword2
```

```
oneTryLeft:
    li $v0, 4                   # load syscall service print_string
    la $a0, oneTry              # load syscall argument for print_string
    syscall                     # make syscall to print_string
    j getPassword2              # jump to getPassword2

noTriesLeft:
    li $v0, 4                   # load syscall service print_string
    la $a0, noTry               # load syscall argument for print_string
    syscall                     # make syscall to print_string
    j exit                      # jump to getPassword2

goodSetup:
    li $v0, 4                   # load syscall service print_string
    la $a0, goodPass            # load syscall argument for print_string
    syscall                     # make syscall to print_string
    j exit                      # jump to exit

exit:
    li $v0, 10                  # load syscall service end_program
    syscall                     # make syscall to end_program
```

Output for Assignment 5

Console

```
Set a password: aaaaaa
Re-enter the password: bbbbbb
Incorrect, you have 2 more chances! Please re-enter the password: Re-enter the password: abbaaa
Incorrect, you have 1 more chance! Please re-enter the password: Re-enter the password: aAaaaa
Incorrect, you have no more chances! Password setup has failed.
```

Console

```
Set a password: Lab
Failed. Please enter a password with the size of 6 to 10!
Set a password: Lablab
Re-enter the password: lab
Incorrect, you have 2 more chances! Please re-enter the password: Re-enter the password: Lablab
Password is setup.
```

Console

```
Set a password: aaaaaa
Re-enter the password: aaaaaa
Password is setup.
```

Pseudo-code used for Assignment 6

```
public static String userInput;
public static String binaryValue = "";
public static void main(String[] args) {
    getUserInput();
}

public static void getUserInput(){
    Scanner input = new Scanner(System.in);
    System.out.print("Hex string: ");
    userInput = input.next();
    loop();
}

public static void loop(){
    char[] hexadecimal = userInput.toCharArray();
    boolean goodValue = true;
    for(int i = 0; i < hexadecimal.length; i++){
        if(goodValue && ((hexadecimal[i] >= '0' && hexadecimal[i] <= '9' ) || (hexadecimal[i] >= 'A' && hexadecimal[i] <= 'F'))){
            convertToBinary(hexadecimal[i]);
        }else{
            goodValue = false;
            break;
        }
    }
    if(goodValue){
        System.out.println("Its binary value: " + binaryValue);
    }else{
        System.out.println("Invalid hex string");
    }
}

public static void convertToBinary(char ch){
    if(ch >= 'A'){
        ch -= 55;
    }else{
        ch -= 48;
    }
    System.out.println((int) ch);
    for(int i = 3; i >= 0; i--){
        if(ch - Math.pow(2, i) >= 0){
            ch -= Math.pow(2, i);
            binaryValue += '1';
        }else{
            binaryValue += '0';
        }
    }
    binaryValue += ' ';
}
```

Code used for Assignment 6

```
# Jeffrey Huang
# RUID: 159004687
# NETID: jh1127
# Assignment 6

# Registers:    $t0 -> hex string provided by user
#               $t1 -> temporary byte holder for hex string
#               $t2 -> copy of $t0 address
#               $t3 -> determines if input is good hexadecimal
#               $t4 -> hexadecimal value is a number 0 - 9
#               $t5 -> hexadecimal value is a letter A - F
#               $t6 -> temporary truth value holder
#               $t7 -> temporary truth value holder
#               $t8 -> counter for power
#               $t9 -> factor for division
#               $s0 -> output string in binary of hex
#               $s1 -> copy of original address of $s0

.data
    hex:        .asciiz     "Hex string: "
    binary:     .asciiz     "Its binary value: "
    badHex:     .asciiz     "Invalid hex string"
    zero:       .asciiz     "0"
    one:        .asciiz     "1"
    space:      .asciiz     " "

.text

main:
## Getting user hex string input
    li $v0, 4                   # load syscall service print_string
    la $a0, hex                 # load syscall argument for print_string
    syscall                     # make syscall for print_string
    li $v0, 8                   # load syscall service read_string
    syscall                     # make syscall for read_string
    move $t0, $a0               # moving user input to register $t0
    move $t2, $t0                   # copying original address to register $t1
    j check                     # jump to check

check:
    lb $t1, 0($t0)              # load byte from address to register $t1
    beq $t1, 10, startConvert       # if $t1 == 0, branch to startConvert
    add $t0, 1                  # increment address by 1
    sge $t6, $t1, '0'           # ch >= '0'
    sle $t7, $t1, '9'           # ch <= '9'
    and $t4, $t6, $t7           # (ch >= '0' && ch <= '9')
    sge $t6, $t1, 'A'           # ch >= 'A'
    sle $t7, $t1, 'F'           # ch <= 'F'
    and $t5, $t6, $t7           # (ch >= 'A' && ch <= 'F')
    or $t3, $t4, $t5            # (ch >= '0' && ch <= '9') || (ch >= 'A' && ch <= 'F')
    beqz $t3, badExit           # if $t3 == 0, branch to badExit
    j check                     # jump to check
```

```
startConvert:
    move $t0, $t2               # putting original address back to register $t0
    li $v0, 4                   # load syscall service print_string
    la $a0, binary              # load syscall argument for print_string
    syscall                     # make syscall for print_string
    j convert                   # jump to convert

convert:
    move $t4, $zero             # clearing register
    move $t5, $zero             # clearing register
    lb $t1, 0($t0)              # load byte from address to register $t1
    beq $t1, 10, goodExit       # if $t1 == 10, branch to goodExit
    add $t0, 1                  # increment address by 1
    li $t8, 3                   # counter = 3
    sge $t6, $t1, '0'           # ch >= '0'
    sle $t7, $t1, '9'           # ch <= '9'
    and $t4, $t6, $t7           # (ch >= '0' && ch <= '9')
    beqz $t4, convertChar       # if $t4 == 0, branch to convertChar
    sge $t6, $t1, 'A'           # ch >= 'A'
    sle $t7, $t1, 'F'           # ch <= 'F'
    and $t5, $t6, $t7           # (ch >= 'A' && ch <= 'F')
    beqz $t5, convertNum        # if $t5 == 0, branch to convertNum


convertNum:
    sub $t1, $t1, 48            # ascii offset for 0- 9
    j convertToBin              # jump to convertToBin

convertChar:
    sub $t1, $t1, 55            # ascii offset for A - F
    j convertToBin              # jump to convertToBin

convertToBin:
    bltz $t8, printSpace        # if $t8 < 0, branch to convert
    sub $t8, $t8, 1             # $t8--
    move $t4, $t8               # load counter to temporary counter
    li $t9, 1                   # initializing factor
    j factor                    # jump to factor

factor:
    bltz $t4, checkFactor       # if $t4 < 0, branch to checkFactor
    sub $t4, $t4, 1             # $t4--
    mul $t9, $t9, 2             # 2 ^ n
    j factor                    # jump to factor

checkFactor:
    bgt $t1, $t9, printOne      # if $t1 > $t9, branch to printOne
    blt $t1, $t9, printZero     # if $t1 < $t9, branch to printZero
    beq $t1, $t9, printOne      # if $t1 = $t9, branch to printOne

printOne:
    sub $t1, $t1, $t9           # $t1 = $t1 - $t9
    li $v0, 4                   # load syscall service print_string
    la $a0, one                 # load syscall argument for print_string
    syscall                     # make syscall for print_string
    j convertToBin              # jump to convertToBin
```

```
factor:
    bltz $t4, checkFactor       # if $t4 < 0, branch to checkFactor
    sub $t4, $t4, 1             # $t4--
    mul $t9, $t9, 2             # 2 ^ n
    j factor                    # jump to factor

checkFactor:
    bgt $t1, $t9, printOne      # if $t1 > $t9, branch to printOne
    blt $t1, $t9, printZero     # if $t1 < $t9, branch to printZero
    beq $t1, $t9, printOne      # if $t1 = $t9, branch to printOne

printOne:
    sub $t1, $t1, $t9           # $t1 = $t1 - $t9
    li $v0, 4                   # load syscall service print_string
    la $a0, one                 # load syscall argument for print_string
    syscall                     # make syscall for print_string
    j convertToBin              # jump to convertToBin

printZero:
    li $v0, 4                   # load syscall service print_string
    la $a0, zero                # load syscall argument for print_string
    syscall                     # make syscall for print_string
    j convertToBin              # jump to convertToBin

printSpace:
    li $v0, 4                   # load syscall service print_string
    la $a0, space               # load syscall argument for print_string
    syscall                     # make syscall for print_string
    j convert                   # jump to convert

badExit:
    li $v0, 4                   # load syscall service print_string
    la $a0, badHex              # load syscall argument for print_string
    syscall                     # make syscall for print_string
    li $v0, 10                  # load syscall service end_program
    syscall                     # load syscall for end_program

goodExit:
    li $v0, 10                  # load syscall service end_program
    syscall                     # make syscall for end_program
```

Output for Assignment 6

Console

```
Hex string: 12345F
Its binary value: 0001 0010 0011 0100 0101 1111
```

Console

```
Hex string: 398723R
Invalid hex string
```

Pseudo-code used for Assignment 7

```
    System.out.println("Please enter the number of input that you would like to enter: ");
    int size = input.nextInt();
    int[] inputs = new int[size];
    for(int i = 0; i < size; i++){
        System.out.println("Please enter a number: ");
        inputs[i] = input.nextInt();
    }
    inputs = bubbleSort(inputs);
    System.out.print("Sorted Array: ");
    for(int i = 0; i < size; i++){
        System.out.print(inputs[i] + " ");
    }
    System.out.println();
    getMedian(inputs, size);
    countPositives(inputs);
}
public static int[] bubbleSort(int numbers[]){
    int temp;
    for(int i = 0; i < numbers.length; i++){
        for(int j = 1; j < (numbers.length -i); j++){
            if(numbers[j-1] > numbers[j]){
                temp = numbers[j-1];
                numbers[j-1]=numbers[j];
                numbers[j]=temp;
            }
        }
    }
    return numbers;
}
public static void getMedian(int numbers[], int size){
    if(size % 2 == 0){
        System.out.println("The median is: " + (numbers[(int) (size / 2) - 1] + numbers[(int) (size / 2)]) / 2 + "\n");
    }else{
        System.out.println("The median is: " + numbers[(int) size/2] + "\n");
    }
}
public static void countPositives(int numbers[]){
    int positives = 0, negatives = 0, zeroes = 0;
    for(int i = 0; i < numbers.length; i++){
        if(numbers[i] == 0){
            zeroes++;
        }else if(numbers[i] < 0){
            negatives++;
        }else{
            positives++;
        }
    }
    System.out.println("The number of positives is: " + positives);
    System.out.println("The number of negatives is: " + negatives);
    System.out.println("The number of zeroes is: " + zeroes);
}
```

Code used for Assignment 7

```
# Jeffrey Huang
# RUID: 159004687
# NETID: jh1127
# Assignment 7

# Registers:    $t0 -> array index counter
#               $t1 -> temporary array base address
#               $t2 ->
#               $t3 -> temporary array base address
#               $t4 ->
#               $t5 ->
#               $t6 ->
#               $t7 ->
#               $t8 ->
#               $t9 ->
#               $s0 -> Array1 base location
#               $s1 -> array size (abitrary value)
#               $s2 -> median value
#               $s3 -> negatives counter
#               $s4 -> positives counter
#               $s5 -> zeroes counter

.data
#   Array1: .word 12, 2, -4, 16, 5, -20, 0, 10, 0xF
    Array1: .space 400                          # array size allows to hold 100 words
    arraySize: .asciiz "Enter size of array: "
    array: .asciiz "Sorted Array: "
    median: .asciiz "\nThe median is: "
    positives: .asciiz "\n\nThe number of positives is: "
    negatives: .asciiz "\nThe number of negatives is: "
    zeroes: .asciiz "\nThe number of zeroes is: "
    space: .asciiz "  "

.text
main:
    la $s0, Array1
    move $t1, $s0                               # moving array base address to $t1
    li $s1, -1                                  # setting initial size = -1
    li $v0, 4                                   # loading syscall service for print_string
    la $a0, arraySize                           # loading syscall argument for print_string
    syscall                                     # making syscall for print_string
    li $v0, 5                                   # loading syscall service for read_int
    syscall                                     # making syscall for read_int
    beq $v0, $0, exit                           # if $v0 == 0, branch to exit
    addi $s1, $v0, -1                           # size-- for index adjustment
## Getting user input
input:
    bgt $t0, $s1, endInput                      # if counter == size, branch to endInput
    li $v0, 5                                   # loading syscall service for read_int
    syscall                                     # making syscall for read_int
    sw $v0, ($t1)                               # storing input into array
    addi $t0, $t0, 1                            # incrementing the array index counter
    addi $t1, $t1, 4                            # incrementing the address by 4 for word
    j input                                     # jump to input
```

```
endInput:
    li $v0, 0xF                         # ending marker of 0xF
    sw $v0, ($t1)                       # storying input into the array
    la $s0, Array1                      # loading base address from Array1
    li $t0, 0                           # changing the index counter to 0

## Bubblesort algorithm
outerLoop:
    beq $t0, $s1, calculate             # if $t0 == $s1, branch to calculate
    addi $t0, $t0, 1                    # array index ++
    li $t1, 0                           # changing base address to 0
    move $t3, $s0                       # moving the base array address to $t3

innerLoop:
    beq $t1, $s1, outerLoop             # if $t1 == $s1, branch to outerLoop
    lw $t4, ($t3)                       # loading first element to $t4
    lw $t5, 4($t3)                      # loading second element to $t5
    addi $t1, $t1, 1                    # incrementing array counter by 1
    blt $t5, $t4, swap                  # if $t5 < $t4, branch to swap
    addi $t3, $t3, 4                    # increment the counter by 4 for word
    j innerLoop                         # jump to innerLoop

swap:
    sw $t5, ($t3)                       # storing the second element as first element
    sw $t4, 4($t3)                      # storing the first element as second element
    addi $t3, $t3, 4                    # increment the counter by 4 for word
    j innerLoop                         # jumpt to innerLoop

calculate:
    li $t0, 2                           # loading division for even numbers
    rem $t1, $s1, $t0                   # if $s1 % 2 == 1 -> even
    bnez $t1, evenSize                  # if $t1 == 1, branch to evenSize
    beqz $t1, oddSize                   # if $t1 == 0, branch to oddSize

oddSize:
    div $t1, $s1, $t0                   # finding the mid point of the array
    mul $t1, $t1, 4                     # multiply by 4 (word factor size)
    move $t2, $s0                       # move the middle element to $t2
    add $t2, $t2, $t1                   # $t2 = $t2 + $t1
    lw $s2, ($t2)                       # load into $s0 for median value
    j negativeCounter                   # jump to negativeCounter

evenSize:
    div $t1, $s1, $t0                   # finding the mid point of the array
    mul $t1, $t1, 4                     # multiply by 4 (word factor size)
    move $t2, $s0                       # moving address to $t2
    add $t2, $t2, $t1                   # adding 4 to the address to get next address
    lw $t0, ($t2)                       # loading lower middle element
    lw $t1, 4($t2)                      # loading higher middle element
    add $t0, $t0, $t1                   # lower middle element + higher middle element
    li $t1, 2                           # loading 2 for division purposes
    div $s2, $t0, $t1                   # dividing sum of lower and higher middle element by 2
    j negativeCounter                   # jump to negativeCounter
```

```
negativeCounter:
    move $t0, $s0                       # move array base address to $t0
    li $s6, -1                          # setting base to recognize the negatives
    addi $t1, $t1, -4                   # subtracting base array address by 4

negativeLoop:
    lw $t1, ($t0)                       # loading first element to register $t1
    bgt $t1, $s6, positiveCounter       # if $t1 > $s6, branch to positiveCounter
    addi $s3, $s3, 1                    # negative++
    addi $t0, $t0, 4                    # incrementing array address by 4
    j negativeLoop                      # jump to negativeLoop

positiveCounter:
    move $t0, $s0                       # load array base address to $t0
    li $s6, 1                           # setting base to recognize the positives
    mul $t1, $s1, 4                     # incrementing array address by 4
    add $t0, $t0, $t1                   # adding top of address to base of address

positiveLoop:
    lw $t1, ($t0)                       # loading last element to register $t0
    blt $t1, $s6, zeroCounter           # if $t1 < $s6, branch to zeroCounter
    addi $s4, $s4, 1                    # positive++
    addi $t0, $t0, -4                   # decrementing array address by 4
    j positiveLoop                      # jump to positiveLoop

zeroCounter:
    sub $s5, $s1, $s3                   # total - negatives = zeroes + positives
    sub $s5, $s5, $s4                   # zeores + positives - positives = zeroes
    addi $s5, $s5, 1                    # adding 1 to adjust for size change
    j printArrayHeader                      # jump to printArrayHeader

printArrayHeader:
    li $v0, 4                           # loading syscall service for print_string
    la $a0, array                       # loading syscall argument for print_string
    syscall                             # making syscall for print_string
    j printArray                        # jump to printArray

printArray:
    blt $s1, $0, printMedian            # if $s1 < 0, branch to printMedian
    lb $a0, ($s0)                       # load byte from sorted array
    li $v0, 1                           # loading syscall service for print_integer
    syscall                             # making syscall for print_integer
    li $v0, 4                           # loading syscall service for print_string
    la $a0, space                       # loading syscall argument for print_string
    syscall                             # making syscall for print_integer
    addi $s0, $s0, 4                    # incrementing the array address by word factor (4)
    addi $s1, $s1, -1                   # decrementing array size by 1
    j printArray                        # jump to printArray

printMedian:
    li $v0, 4                           # loading syscall service for print_string
    la $a0, median                      # loading syscall argument for print_string
    syscall                             # making syscall for print_string
    li $v0, 1                           # loading syscall service for print_integer
    move $a0, $s2                       # loading syscall argument for print_integer
    syscall                             # making syscall for print_integer
    j printPositives                    # jump to printPositives
```

```
printNegatives:
    li $v0, 4                           # loading syscall service for print_string
    la $a0, negatives                   # loading syscall argument for print_string
    syscall                             # making syscall for print_string
    li $v0, 1                           # loading syscall service for print_integer
    move $a0, $s3                       # loading syscall argument for print_integer
    syscall                             # making syscall for print_integer
    j printZeroes                       # jump to printZeroes

printPositives:
    li $v0, 4                           # loading syscall service for print_string
    la $a0, positives                   # loading syscall argument for print_string
    syscall                             # making syscall for print_string
    li $v0, 1                           # loading syscall service for print_integer
    move $a0, $s4                       # loading syscall argument for print_integer
    syscall                             # making syscall for print_integer
    j printNegatives                    # jump to printNegatives

printZeroes:
    li $v0, 4                           # loading syscall service for print_string
    la $a0, zeroes                      # loading syscall argument for print_string
    syscall                             # making syscall for print_string
    li $v0, 1                           # loading syscall service for print_integer
    move $a0, $s5                       # loading syscall argument for print_integer
    syscall                             # making syscall for print_integer
    j exit                              # jump to exit

exit:
    li $v0, 10                          # loading syscall service for end_program
    syscall                             # making syscall for end_program
```

Outputs for Assignment 7

Console

```
Enter size of array: 8
-20
-4
0
2
5
10
12
16
Sorted Array: -20  -4  0  2  5  10  12  16
The median is: 3

The number of positives is: 5
The number of negatives is: 2
The number of zeroes is: 1
```

Console

```
Enter size of array: 10
1
2
3
4
5
6
7
8
9
10
Sorted Array: 1  2  3  4  5  6  7  8  9  10
The median is: 5

The number of positives is: 10
The number of negatives is: 0
The number of zeroes is: 0
```