**MID-TERM REPORT**

**IT3280E– 152049– Assembly Language and Computer Architecture Lab**

**Type A – Integer Number:**

*8. Enter a positive integer N (with at least 3 digits) from the keyboard. Print out the largest digit of N.*

* Understanding the Problem:

+ Read a number from user input

+ Breakdown the number digit by digit

+ Compare each digit to find the maximum digit

+ Print the largest digit as output

* Implementation:

We use a loop to iterate through each digit of *N*. In each loop we take the result of *N* divide by 10 to get the last digit. Then we compare the said digit with the current largest. If the taken digit is greater than the current largest, it updates the largest digit. After checking the last digit, the code removes it by integer-dividing *N* by 10, shifting the digit to the right. Continue this process until *N* becomes zero (all digits have been processed).

* Source code:

|  |
| --- |
| .data  msg: .asciz "N: "  newline: .asciz "Output: "  .text  # Registers  # t0 = integer  # t1 = current largest digit  # t2 = last digit  main:  # prompt input  li a7, 4  la a0, msg  ecall    # read integer input  li a7, 5  ecall  mv t0, a0    # initialize  li t1, 0  li t3, 10 # for modulus operation    find:  beqz t0, res    # extract the last digit  rem t2, t0, t3  div t0, t0, t3    # compare and update largest digit  bge t2, t1, larg  j find # skip if current is larger    larg:  mv t1, t2 # update  j find    res:  # print result  li a7, 4  la a0, newline  ecall    addi a0, t1, 48  li a7, 11 # print\_char  ecall    li a7, 10  ecall |

* Example input/output:





A close up of numbers

Description automatically generated

**Type B - Arrays:**

*18. Enter an array of integers from the keyboard. Print out the smallest odd number that is greater than all the even numbers in the array.*

* Understanding the Problem:

+ Read an array of integers from user input (process multiple values)

+ First check if number is even or odd

+ Determine the largest even number in array by iterating through the array

+ Find smallest odd number that is greater than this largest even number

+ If there are none, programm prints out “None”

* Implementation:

We accept user-inputed array of integers by first asking user the array size and then take input for each element one at a time using a loop. For each element of the array, the code checks if it’s odd or even. If the number is odd, it compares it to the max\_int and update if the current odd number is smaller. If the number is even, it compares it to min\_int and update if the current even number is larger. After iterating through the loop to process all elements, it checks if the smallest odd number is greater than the largest even number. If yes, programm prints the result, else prints “None”

* Source code:

|  |
| --- |
| .data  msg1: .asciz "Input: "  msg2: .asciz "Output: "  msg3: .asciz "None"  newline: .asciz "\n"  .text  # Registers:  # t1 = smallest odd number  # t2 = largest even number  # t3 = number of elements  # t4 = current element  # t5 = check if number is odd or even  main:  # prompt input number of elements  li a7, 4  la a0, msg1  ecall    # read the number of elements  li a7, 5  ecall  mv t0, a0    # initialize t1 to a large value and t2 to a small value  li t1, 2147483647 # initialized to max int  li t2, -2147483647 # initialized to min int    # initialize loop counter  li t3, 0  li t6, 2 # for modulus operation  loop:  bge t3, t0, res # if index >= number of elements, go to result check    # read the next array element  li a7, 5 # read\_int  ecall  mv t4, a0    # check if the element is odd or even  rem t5, t4, t6 # t5 = t4 % 2  beq t5, zero, even # if t4 is even, go to even    # check if odd < t1  blt t4, t1, odd  j even # skip to even check if no update  odd:  mv t1, t4 # update smallest odd to t4  j even # go to even check  even:  # check if larger  beq t5, zero, check\_even  j next # if not even, skip  check\_even:  ble t4, t2, next # if t4 <= largest even, skip update  mv t2, t4 # update t4  next:  addi t3, t3, 1 # increment index counter  j loop # repeat the loop  res:  # check if t1 > t2  ble t1, t2, none # if yes, go to none  # print the smallest odd number  li a7, 4  la a0, msg2  ecall  mv a0, t1    li a7, 1 # print\_int  ecall  j end  none:  li a7, 4  la a0, msg3  ecall  end:  li a7, 10  ecall |

* Example input/output:

A white background with black text

Description automatically generated

A white background with black text

Description automatically generated

**Type C - Strings:**

*14. Enter two strings A and B, print out the uppercase letters that appear in both A and B.*

* Understanding the problem:

+ Read two strings A and B from user and store the inputs in memory

+ Check each letter from string A for uppercase letter and compare it with each character in string B

+ If matches, print that character

* Implementation:

After user inputs the two strings, we stores them in two buffer and initializes two registers to represent the ASCII range for uppercase letters, 'A' to 'Z'. We use two pointers to iterate through each pair of. The first pointer points to the first character of string A, the second pointer points to the first character of string B. The programm then iterates through each character in string A to check if it falls within the uppercase range. For each uppercase character found in A, the program scans B to see if the said character also exists there. The process continues untill all characters in A have been checked. Prints result if there is.

* Source code:

|  |
| --- |
| .data  inputA: .space 80  inputB: .space 80  msg1: .asciz "String A: "  msg2: .asciz "String B: "  msg3: .asciz "Output: "  msg4: .asciz "None\n"  .text  main:  # prompt input for string A  li a7, 4  la a0, msg1  ecall    # read string A  li a7, 8  la a0, inputA  li a1, 80  ecall    # prompt input for string B  li a7, 4  la a0, msg2  ecall  # read string B  li a7, 8  la a0, inputB  ecall    # print result message  li a7, 4  la a0, msg3  ecall    # initialize A and Z  la t0, inputA  li t2, 'A'  li t3, 'Z'    # track match found  li t6, 0 # no match    uppercase\_A:  lb t1, 0(t0) # load byte from inputA  beq t1, zero, check\_none # if null check if no match found    blt t1, t2, next # if t1 < 'A', go to next  bgt t1, t3, next # if t1 > 'Z', go to next    la t4, inputB # start of inputB    uppercase\_B:  lb t5, 0(t4) # load byte from inputB  beq t5, zero, next # if null move to next char in inputA    # compare t1 and t5 (A and B)  bne t1, t5, skip # if not equal, skip    # if match is found, set flag and print character  li a7, 11 # print\_char  add a0, t1, zero # load character  ecall  li t6, 1 # set match found flag  j next # move to next character in inputA    skip:  addi t4, t4, 1 # move to next character in inputB  j uppercase\_B # continue searching  next:  addi t0, t0, 1 # move to next character in inputA  j uppercase\_A # continue searching  check\_none:  # if no match found, print "None"  beq t6, zero, none  j end    none:  li a7, 4  la a0, msg4 # print "None"  ecall  end:  li a7, 10  ecall |

* Example input/output:

**A black text on a white background

Description automatically generated**

**A black text on a white background

Description automatically generated**

**A black text on a white background

Description automatically generated**