COE 301/ICS 233, Term 172

Computer Architecture & Assembly Language

HW# 3 Solution

Q.1. Write a MIPS assembly program that does the following:

- **(i)** Ask the user to enter the number of rows R and read it.
- **(ii)** Ask the user to enter the number of columns C and read it.
- **(iii)** Ask the user to enter a two-dimensional array of RxC characters. Elements of a single row should be separated by a single space and each row is read in a new line.
- **(iv)** Print the entered array in a new line printing also its entered dimensions.
- **(v)** Ask the user to enter two row numbers
- **(vi)** Exchange the two entered rows and print the array after the exchange.

A sample execution of the program is shown below:

```
Enter number of rows in the array: 3
Enter number of columns in the array: 5
Enter an array of 3x5 characters:
0 1 2 3 4
5 6 7 8 9
a b c d e
The entered 3x5 array is:
0 1 2 3 4
5 6 7 8 9
a b c d e
Enter a row number: 0
Enter another row number: 2
The array after exchanging Row 0 and Row 2 is:
a b c d e
5 6 7 8 9
0 1 2 3 4
```

```
################# Data segment #####################
.data
msg1: .asciiz "Enter number of rows in the array:"
msg2: .asciiz "Enter number of columns in the array:"
msg3: .asciiz "Enter an array of "
msg4: .asciiz " characters:\n"
```

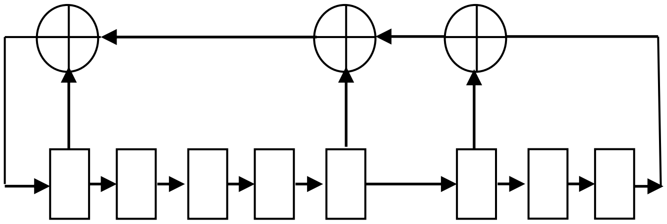
```
msg5: .asciiz "The entered "
msg6: .asciiz " array is:\n"
msg7: .asciiz "Enter a row number:"
msg8: .asciiz "Enter another row number:"
msg9: .asciiz "The array after exchanging Row "
msg10: .asciiz " and Row "
msg11: .asciiz " is:\n"

Array: .space 400# it is assumed that we have a max. array size of 20x20
Buffer: .space 40
################# Code segment #####################
.text
.globl main
main:       # main program entry

# Read number of rows and columns

    la $a0, msg1
    li $v0, 4
    syscall
    li $v0, 5
    syscall
    move $s0, $v0 # Number of rows

    la $a0, msg2
    li $v0, 4
    syscall
    li $v0, 5
    syscall
    move $s1, $v0 # Number of columns

# Printing messgae for reading the array

    la $a0, msg3
    li $v0, 4
    syscall
    move $a0, $s0
    li $v0, 1
    syscall
    li $a0,'x'
    li $v0, 11
    syscall
    move $a0, $s1
    li $v0, 1
    syscall
    la $a0, msg4
    li $v0, 4
    syscall
```

```
# Reading the array and storing it

    la $s2, Array
    move $t0, $s0
NextR:
    la $a0, Buffer
    move $a1, $s1
    sll $a1, $a1, 1
    addi $a1, $a1, 1
    li $v0, 8
    syscall
    move $t1, $s1
NextC:
    lb $t2, ($a0)
    sb $t2, ($s2)
    addi $a0, $a0, 2
    addi $s2, $s2, 1
    addi $t1, $t1, -1
    bnez $t1, NextC

    addi $t0, $t0, -1
    bnez $t0, NextR

# Displaying the entered array

    la $a0, msg5
    li $v0, 4
    syscall
    move $a0, $s0
    li $v0, 1
    syscall
    li $a0,'x'
    li $v0, 11
    syscall
    move $a0, $s1
    li $v0, 1
    syscall
    la $a0, msg6
    li $v0, 4
    syscall
    la $a0, Array
    move $a1, $s0
    move $a2, $s1
    jal DispArray
```

```
# Getting First Row
    la $a0, msg7
    li $v0, 4
    syscall
    li $v0, 5
    syscall
    move $s3, $v0 # First row to be exchanged

# Getting Second Row
    la $a0, msg8
    li $v0, 4
    syscall
    li $v0, 5
    syscall
    move $s4, $v0 # Second row to be exchanged

# Exchanging the two rows
    la $a0, Array
    move $a1, $s1
    move $a2, $s3
    move $a3, $s4
    jal ExchgRows

# Displaying the array after the exchange

    la $a0, msg9
    li $v0, 4
    syscall
    move $a0, $s3
    li $v0, 1
    syscall
    la $a0, msg10
    li $v0, 4
    syscall
    move $a0, $s4
    li $v0, 1
    syscall
    la $a0, msg11
    li $v0, 4
    syscall

    la $a0, Array
    move $a1, $s0
    move $a2, $s1
    jal DispArray

    li $v0, 10        # Exit program
    syscall
```

```
# A procedure that receives in $a0 the address of an array,
# in $a1 the number of rows, and in $a2 the number of columns
# and displays the array.
DispArray:
    move $t0, $a0
NR:
    move $t1, $a2
NC:
    lb $a0, ($t0)
    li $v0, 11
    syscall
    li $a0, ' '
    syscall
    addi $t0, $t0, 1
    addi $t1, $t1, -1
    bnez $t1, NC
    li $a0, 10
    syscall
    addi $a1, $a1, -1
    bnez $a1, NR
    jr $ra

# A procedure that receives in $a0 the address of an array,
# in $a1 the number of columns, and in $a2 & $a3 the two
# rows to be exchanged.
ExchgRows:

# Compute starting address of first row
    mul $t0, $a1, $a2
    add $t0, $t0, $a0
# Compute starting address of second row
    mul $t1, $a1, $a3
    add $t1, $t1, $a0
# Performing the exchange
Next:
    lb $t3, ($t0)
    lb $t4, ($t1)
    sb $t3, ($t1)
    sb $t4, ($t0)
    addi $t0, $t0, 1
    addi $t1, $t1, 1
    addi $a1, $a1, -1
    bnez $a1, Next
    jr $ra
```

Q.2. You are required to write a MIPS assembly program to implement a pseudo random generator using Liner Feedback Shift Register (LFSR). An example of an 8-bit LFSR is shown below:



Two important characteristics of an LFSR are the Feedback Polynomial, which determines the flip-flops (FFs) that are XORed to compute the shifted bit, and the seed which determines the initial content of the FFs. Depending on the Feedback polynomial, the LFSR can generate a maximal-length sequence without repetition, or it may not. The seed can be any number other than 0.

The 8-bit LFSR shown above is a maximal-length i.e. it is guaranteed to generate a random sequence in the range from 1 to 255 before it repeats again.

The Feedback polynomial for the above LFSR can be represented as 10001101. Note that 1 indicates that there is feedback connection, while 0 indicates that there is no feedback connection.

(i) Write a procedure **READB** to read a binary number and store it in $v0. The procedure should report an error message and ask the user to reenter the number if the decimal value of the entered number is 0. Also, the user does not have to enter the whole 8-bits. If he enters less than 8 bits and hits return the remaining most significant bits should be assumed 0. Also, if any digit entered is other than a binary digit, an error message should be reported.

(ii) Write a procedure, **RAND8**, that implements an 8-bit pseudo random generator. The procedure should be given the Feedback polynomial, and the seed as parameters in $a0 and $a1 registers and it should generate the next random number in $v0.

(iii) Ask the user to enter an 8-bit feedback ploynomial and an 8-bit seed in binary. Use the procedure READB for this purbose. Then, ask the user to enter a string of characters. Then, encrypt the string using RAND8 as follows. Each character is encrypted by XORing the least significant 4-bits of the ASCII code of the character with the least significant 4 bits and the most significant 4-bits of the generated random number . For example, assume the character to be encrypted is 'A'=0x41 and the random number is 0xA1. Then, the encrypted character will have the ASCII code 0x4A='J'. To decrypt the character, the decrypted character 0x4A='J', will be XORed with the same corresponding random number used for encryption i.e. 0xA1 and this will generate the original character 0x41='A'. As an example, show the encryption of the string **"This is an interesting assignment"**. Then, rerun your program giving it the encrypted string and it should correctly decrypt it to "**This is an interesting assignment**". Encrypt this with the feedback polynomial 10001101 and a seed of 00001111.

```
################# Data segment #####################
.data
msg1: .asciiz "Enter an 8-bit feedback polynomial: "
msg2: .asciiz "Enter an 8-bit seed:"
msg3: .asciiz "Enter a message to encrypt/decrypt: "
msg4: .asciiz "Encrypted/Decrypted message is: "
buffer: .space 101# Assuming maximum message length is 100 characters
bnum: .space 9
Errmsg: .asciiz "One of the entered digits is not a binary digit, renter the number again\n"
Errmsg2: .asciiz "Entered number is zero, renter the number again\n"

################# Code segment #####################
.text
.globl main
main:       # main program entry

# Getting an 8-bit feedback polynomial
    li $v0, 4
    la $a0, msg1
    syscall
    jal ReadB
    move $s0, $v0

# Getting an 8-bit seed
    li $v0, 4
    la $a0, msg2
    syscall
    jal ReadB
    move $s1, $v0

# Reading the message yo be encrypted/decrypted
    li $v0, 4
    la $a0, msg3
    syscall
    li $v0, 8
    la $a0, buffer
    li $a1, 101
    syscall

# Encrypting/Decrypting message
    li $v0, 4
    la $a0, msg4
    syscall
   li $s4, 10
   li $s5, 100                    # Loop counter
    la $s2, buffer
NextChar:
    lb $s3, ($s2)
    beq $s3, $s4, DoneE

# Generating next random number
    move $a0, $s0
    move $a1, $s1
    jal RAND8
    move $s1, $v0                 # Storing returned number as next seed

    move $t2, $v0                 # encrypting/decrypting character
    srl $t2, $t2, 4
    andi $v0, $v0, 15
    xor $v0, $v0, $t2
    xor $s3, $s3, $v0
    sb $s3, ($s2)

    addi $s2, $s2, 1
    addi $s5, $s5, -1
    bnez $s5, NextChar
DoneE:
    li $v0, 4
    la $a0, buffer
    syscall
    li $v0, 10          # Exit program
    syscall

# Procedure for reading an 8-bit binary number
# The read number will be returned in $v0.
ReadB:
# Reading the number as a string
Again:
    xor $v1, $v1, $v1 # to hold binary number
    li $v0, 8
    la $a0, bnum
    move $a2, $a0
    li $a1, 9
    syscall
    li $t0, 8           #loop counter
    li $t2, 10
    li $t3, '0'
    li $t4, '1'
Next:
    lb $t1, ($a2)
   beq $t1, $t2, Done
   sll $v1, $v1, 1
   beq $t1, $t3, Bdigit
    bne $t1, $t4, Error
Bdigit:
   andi $t1, $t1, 1
    or $v1, $v1, $t1
    addi $a2, $a2, 1
   addi $t0, $t0, -1
    bnez $t0, Next
    bnez $v1, Done
    la $a0, Errmsg2
    li $v0, 4
    syscall
    j Again
Error:
    la $a0, Errmsg
    li $v0, 4
    syscall
    j Again
Done:
    move $v0, $v1
    jr $ra

# Procedure for implementing an 8-bit LFSR-based pseudo random
# number generater. The procedure is given the Feedback polynomial,
# and the seed as parameters in $a0 and $a1 registers and it generates
# the next random number in $v0.

RAND8:
    move $v0, $a1
    and $a1, $a1, $a0       #Mask the bits that should not be Xored
# Count number of 1's in $a0
    li $t0, 8                         # Loop counter
    xor $t1, $t1, $t1         # Number of ones
Loop:
    move $t2, $a1
    andi $t2, $t2, 1
    add $t1, $t1, $t2
    srl $a1, $a1, 1
    addi $t0, $t0, -1
    bnez $t0, Loop
    srl $v0, $v0, 1
   andi $t1, $t1, 1 # Check if number of ones is even or odd
    beqz $t1, Skip
    ori $v0, 0x0080
Skip:
    jr $ra
```

Snapshot of running the program is given below**:**

```
Enter an 8-bit feedback polynomial: **** user input : 10001101
Enter an 8-bit seed:**** user input : 00001111
Enter a message to encrypt/decrypt: **** user input : This is an interesting assignment
Encrypted/Decrypted message is: [gnx%cv"`n(e`sf{i}{nmf i • ubjhnlbr


Enter an 8-bit feedback polynomial: **** user input : 10001101
Enter an 8-bit seed:**** user input : 00001111
Enter a message to encrypt/decrypt: **** user input : [gnx%cv"`n(e`sf{i}{nmf i • ubjhnlbr
Encrypted/Decrypted message is: This is an interesting assignment
```