**COMP2611 Fall 2019 Homework #2**

**(Due 11:55PM, Monday, November 4, 2019)**

**Notes:**

* The deadline of this homework is at 11:55pm on Monday, 4 November 2019.
* Write your code in given MIPS assembly skeleton files. Add your own code under TODOs in the skeleton code. Keep other parts of the skeleton code unchanged.
* Make procedure calls with the registers as specified in the skeleton, otherwise the provided procedures may not work properly. Preserve registers according to the MIPS register convention on slide 76 of the ISA note set.
* Zip the three finished MIPS assembly files into a single zip file, <*your\_stu\_id>.zip* file (without the brackets). Do not change names of the given skeleton files.
* To submit, first find the Canvas page of COMP2611, homework2, and then upload the “<*your\_stu\_id>.zip*” file. You can upload for as many number of times as you like, only the last one before the deadline will be marked.
* Solutions of this homework will be posted at the course web right after the deadline, so **no late submission will be accepted.**
* **Your submitted program must be able to run under MARS, otherwise it will not be marked.**

**Question 1: Counting Integers in an Array (20 marks)**

The C++ program given below gets 10 user integer inputs, and stores the integers into the array A[]. Then the program will run the binCount() function to count the number of occurrences of the various inputted integers in A[], and store the results to the array A\_count[]. The C++ program will output the count results in A\_count[]. Refer to the C++ program below for the details. The partial MIPS implementation of the C++ program is in the skeleton file, *Count.asm*, provided. Your job is to refer to the C++ program and implement the bincount() function in MIPS, so that it works like the bincount() function of the C++ program. Write your code under the TODO. You can assume the user always inputs valid integers in the interval [0, 99]. You should submit the solution using the original file name *Count.asm*, do not change the code other than adding the TODO.

The sample run of the MIPS assembly program is provided below for your reference.

**A sample run of the MIPS program:**

Please enter an integer from 0 to 99 into the array A[] one by one:

A[0]: 2

A[1]: 2

A[2]: 2

A[3]: 2

A[4]: 15

A[5]: 15

A[6]: 15

A[7]: 15

A[8]: 15

A[9]: 15

Original array:2 2 2 2 15 15 15 15 15 15

Bin count output:

The count of 2 in A[] is: 4

The count of 15 in A[] is: 6

-- program is finished running --

**C++ Program for your reference:**

**#include**<iostream>

**using** **namespace** std;

**void** **arrayPrint**(**const** **int** array[], **int** size);

**void** **countPrint**(**const** **int** array[], **int** size);

**void** **binCount**(**const** **int** A[], **int** A\_count[], **int** size);

**int** **main**(){

**const** **int** SIZE = 10;

**const** **int** MAX\_RANGE= 99;

**int** A[SIZE];

**int** A\_count[MAX\_RANGE+1]={}; // empty initialization list will initialize

// every element of A\_count[] to 0

cout << " Please enter an integer from 0 to 99 into the array A[] one by one:"<<**endl**;

//assume user input is always valid

**for**(**int** i=0;i<SIZE;i++){

cout << "Array["<<i<<"]: ";

cin >> A[i];

}

binCount(A,A\_count,SIZE);

cout << "Original array:";

arrayPrint(A,SIZE);

cout << "Bin count output:"<<**endl**;

countPrint(A\_count,MAX\_RANGE+1);

**return** 0;

}

**void** **binCount**(**const** **int** A[], **int** A\_count[], **int** size){

**for**(**int** i=0;i<size;i++)

A\_count[A[i]]= A\_count[A[i]]+1;

}

**void** **arrayPrint**(**const** **int** array[],**int** size){

**for**(**int** i=0;i<size;i++)

cout<<array[i]<<'\t';

cout<<**endl**;

}

**void** **countPrint**(**const** **int** array[],**int** size){

**for**(**int** i=0;i<size;i++)

**if**(array[i]>0)

{cout<<"The count of "<<i<<" in A[] is:"<<'\t';

cout<<array[i]<<**endl**;}

cout<<**endl**;

}

**Question 2: 2-by-2 Matrix Multiplication (20 marks)**

Given *2-by-2* matrices, A and B:

|  |  |
| --- | --- |
|  |  |

The product of the two matrices, C=AB, is defined as follows:

Where:

Assume the matrix A and matrix B are physically stored in the 1-D arrays *MatrixA[]* and *MatrixB[]* by row order in MIPS,

*origin𝑀𝑎𝑡𝑟𝑖𝑥A[]={, , , }*

*origin𝑀𝑎𝑡𝑟𝑖𝑥B[]={, , , }*

The matrix C is stored in another 1-D array *outputMatrix[]* in MIPS:

*outputMatrix[]={, , , }*.

Given matrix A=, and B=, then C= and therefore *outputMatrix[] = {1, 1, 5, 5}* in MIPS

Refer to the C++ program below and implement the multiplyMatrix()function in the MIPS skeleton code, *MultiplyMatrix.asm*, so that it works like the multiplyMatrix()function of the C++ program. Write your code under the TODO. Use the multiply function supplied in the MIPS skeleton code to calculate for multiplications. You should submit the solution using the original file name *MultiplyMatrix.asm*, do not change the code other than adding the TODO.

The sample run of the MIPS assembly program is provided below for your reference.

**A sample run of the MIPS program:**

MatrixA[0]: 0

MatrixA[1]: 1

MatrixA[2]: 2

MatrixA[3]: 3

MatrixB[0]: 1

MatrixB[1]: 1

MatrixB[2]: 1

MatrixB[3]: 1

The original matrix A is

0 1

2 3

The original matrix B is

1 1

1 1

The multiplied output matrix is

1 1

5 5

-- program is finished running --

**C++ Program for your reference:**

**#include**<iostream>

**#include**<ctime>

**using** **namespace** std;

**void** **matrixMultiply**(**int** a[], **int** b[], **int** c[], **int** n, **int** col);

**void** **print**(**int** a[], **int** row, **int** col);

**int** **main**() {

**int** n = 4;

**int** col = 2, row = 2; // col/row in the matrix

**int** originalMatrixA[4];

**int** originalMatrixB[4];

**int** outputMatrix[4];

//assume user input is always valid

**for**(**int** i=0;i<n;i++){

cout << "MatrixA["<<i<<"]: ";

cin >> originalMatrixA[i];

}

**for**(**int** i=0;i<n;i++){

cout << "MatrixB["<<i<<"]: ";

cin >> originalMatrixB[i];

}

cout << "The original matrix A is\n";

print(originalMatrixA, row, col);

cout << "The original matrix B is\n";

print(originalMatrixB, row, col);

matrixMultiply(originalMatrixA, originalMatrixB, outputMatrix, n, col);

cout << "The multiplied output matrix is\n";

print(outputMatrix, col, row);

**return** 0;

}

**void** **matrixMultiply**(**int** a[], **int** b[], **int** c[], **int** n, **int** col) {

// use the multiply function in MIPS to replace the \* operator below

c[0] = a[0]\*b[0] + a[1]\*b[2];

c[1] = a[0]\*b[1] + a[1]\*b[3];

c[2] = a[2]\*b[0] + a[3]\*b[2];

c[3] = a[2]\*b[1] + a[3]\*b[3];

}

**void** **print**(**int** a[], **int** row, **int** col) {

**for** (**int** i = 0; i < row; i++) {

**for**(**int** j = 0; j < col; j++)

cout << a[i\*col + j] << " ";

cout << **endl**;

}

cout << **endl**;

}

**Question 3: Generating an *n-bit* Gray Code (20 marks)**

The C++ program given below generates and outputs an *n-bit* Gray code, where *n*=1, 2, 3, 4 is a user input value.

To generate *n-bit* Gray code, we can start from a trivial *1-bit* Gray code: 0, 1

With the *1-bit* Gray code we can make a *2-bit* Gray code by the following steps:

1. copy the *1-bit* Gray code, and then do a "mirroring" of the *1-bit* Gray codes to give the sequence: 0, 1, 1, 0,

note that the two *1-bit* values "1, 0" at the right is the result of mirroring "0", "1" at the

left

1. put 0 to the left binary values "0" and "1", making them "00" and "01",
2. put 1 to the left hand side (mirrored) binary values "1"and "0", making them "11" and "10",
3. we now have a 2-bit Gray code: 00 01 11 10

Using exactly the same steps on a *2-bit* Gray code, we can generate a *3-bit* Gray code. In general, we can make an *n-bit* Gray code from an *(n-1)-bit* Gray code.

Your job is to refer to the C++ program below, implement the mirror() function and the addOneToLeft()function in the MIPS skeleton code, *GrayCode.asm*, so that they work exactly like the mirror()and the addOneToLeft()functions in the C++ program. Write your code under the TODOs. You should submit the solution using the original file name *GrayCode.asm*, do not change the code other than adding the TODOs.

The sample run of the MIPS assembly program is provided below for your reference.

**A sample run of the MIPS program:**

Please enter the bit number(range:1~4) of Gray Code: 4

The Gray Code output is

0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111 1110 1010 1011 1001 1000

-- program is finished running –

**C++ Program for your reference:**

**#include**<iostream>

**#include**<string>

**#include** <math.h>

**using** **namespace** std;

**void** **mirror**(**int** n, **int** smallArray[], **int** bigArray[]);

**void** **addOneToLeft**(**int** n, **int** array[]);

**void** **printGrayCode**(**int** n, **int** array[]);

**int** **main**(){

**int** n;

//array1[] holds the trivial 1-bit Gray code

**int** array1[2]={0,1};

//array2[] holds the 2-bit Gray code generated from the Gray code in array1

**int** array2[4];

//array3[] holds the 3-bit Gray code generated from the Gray code in array2

**int** array3[8];

//array4[] holds the 4-bit Gray code generated from the Gray code in array3

**int** array4[16];

cout << "Please enter the bit number (range: 1~4) of Gray Code: ";

cin>>n;

cout<<"The Gray Code output is"<<**endl**;

// assume user entered n is always valid in the range 1,2,3,4

**if** (n == 1){//user indicates he/she wants to output the 1-bit Gray code

// no need to generate, it is already in array1[], so just print it

printGrayCode(n, array1);

}

**else{**

// generate 2-bit Gray Code

mirror(2, array1, array2);

addOneToLeft(2, array2);

**if** (n==2){

//user wants to output the 2-bit Gray code

printGrayCode(n, array2); **return** 0;}

// generate 3-bit Gray Code

mirror(4, array2, array3);

addOneToLeft(4, array3);

**if** (n==3){

//user wants to output the 3-bit Gray code

printGrayCode(n, array3); **return** 0;}

// generate 4-bit Gray Code

mirror(8, array3, array4);

addOneToLeft(8, array4);

**if** (n==4){

// user wants to output the 4-bit Gray code

printGrayCode(n, array4); **return** 0;}

} // end if (n==4){}

} // end else{}

**void** **mirror**(**int** n, **int** smallArray[], **int** bigArray[]){

**int** len=n+n; // length of the bigArray[] for the new Gray code is 2\*n

//mirror the elements of smallArray[] to occupy the right half of

//bigArray[]

**for**(**int** i=len/2;i<len;i++){

bigArray[i]=smallArray[len-i-1];

}

//copy the elements of smallArray[] to occupy the left half of

//bigArray[]

**for**(**int** i=0;i<len/2;i++){

bigArray[i]=smallArray[i];

}

**return**;

}

**void** **addOneToLeft**(**int** n, **int** array[]){

**int** len=n+n;

**int** value=n;

//add one to the elements at the right half of array[]

//leave the elements at the left half of the array[] unchanged

**for**(**int** i=len/2;i<len;i++){

array[i]= value+array[i];

}

**return**;

}

//output each elements in the array[] in n bits.

//pad "0" to the left whenever necessary to make the output to be n-bit

**void** **printGrayCode**(**int** n, **int** array[]){

**for**(**int** i=0;i<(**int**)**pow**(2,n);i++){

**int** r;

**int** q;

**int** ch[n];

**for** (**int** j=n;j>0;j--){

r = array[i] % 2;

q = array[i] / 2;

array[i] = q;

ch[j]=r;

}

**for** (**int** j=1;j<=n;j++){

cout<<ch[j];

}

cout<<" ";

}

**return**;

}